

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

Pentaho Kettle Solutions:
Building Open Source ETL Solutions
with Pentaho Data Integration

Pentaho Kettle 解决方案:

使用PDI构建开源ETL解决方案

Matt Casters
Roland Bouman 著
Jos van Dongen

初建军 译
曹雪梅



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>



中国专业数据分析社区

www.dataguru.cn

炼数成金（www.dataguru.cn）成立于2011年，目标是用新兴的社交网站的形式，把各应用领域的业务专家、数据分析专家、IT专家以及这些领域的从业人员，学习者关联起来，使之能高效率地沟通交流，帮助企业 and 用户能在海量数据中寻找出价值。炼数成金社区现已成为中国数据分析行业从业人员主要的集散地，业内的黄埔军校。欢迎有志者加入我们的行列，共同将其打造为中国最具影响力的大数据分析垂直社交网站。

内容简介

Pentaho Kettle Solutions:
Building Open Source ETL Solutions
with Pentaho Data Integration

Pentaho Kettle

解决方案:

使用PDI构建开源ETL解决方案

Matt Casters
Roland Bouman
Jos van Dongen

著

初建军
曹雪梅

译

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书主要介绍如何使用开源ETL工具来完成数据整合工作。

本书介绍的PDI(Kettle)是一种开源的 ETL 解决方案,书中介绍了如何使用PDI来实现数据的剖析、清洗、校验、抽取、转换、加载等各类常见的ETL类工作。

除了ODS/DW类比较大型的应用外, Kettle 实际还可以为中小企业提供灵活的数据抽取和数据处理的功能。Kettle除了支持各种关系型数据库、HBase、MongoDB这样的NoSQL数据源外,它还支持Excel、Access这类小型的数据源。并且通过插件扩展, Kettle 可以支持各类数据源。本书详细介绍了Kettle可以处理的数据源,而且详细介绍了如何使用Kettle抽取增量数据。

Kettle 的数据处理功能也很强大,除了选择、过滤、分组、连接、排序这些常用的功能外, Kettle 里的Java表达式、正则表达式、Java脚本、Java类等功能都非常灵活而强大,都非常适合于各种数据处理功能。本书也使用了一些篇幅介绍Kettle这些灵活的数据处理功能。

本书后面章节介绍了如何在 Kettle 上开发插件,如何使用Kettle处理实时数据流,以及如何在Amazon AWS上运行Kettle 等一些高级主题。

除了介绍PDI(Kettle)工具的使用和功能,本书还结合Kimball博士的数据仓库和ETL子系统的理论,从实践的角度介绍数据仓库的模型设计、数据仓库的构建方法,以及如何使用 PDI实现Kimball博士提出的34种ETL子系统。

Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration

ISBN: 9780470635179

Original English Edition Copyright © 2010 by Wiley Publishing, Inc.

All rights reserved. This translation published under license.

Authorized Translation of the Edition published by Wiley Publishing, Inc. Indianapolis, Indiana.

No part of this book may be reproduced in any form without the written permission of Wiley Publishing, Inc.

Copies of this book sold without a Wiley sticker on the back cover are unauthorized and illegal.

本书简体中文版专有版权由Wiley Publishing, Inc.授予电子工业出版社。专有版权受法律保护。

本书封底贴有Wiley Publishing, Inc.防伪标签,无标签者不得销售。

版权贸易合同登记号 图字: 01-2014-0738

图书在版编目(CIP)数据

Pentaho Kettle解决方案:使用PDI构建开源ETL解决方案 / (美)卡斯特 (Casters,M.), (美)布曼(Bouman,R.), (美)东恩 (Dongen,J.V.) 著; 初建军, 曹雪梅译. —北京: 电子工业出版社, 2014.3

书名原文: Pentaho Kettle Solutions:Building Open Source ETL Solutions with Pentaho Data Integration

ISBN 978-7-121-22445-4

I. ①P… II. ①卡… ②布… ③东… ④初… ⑤曹… III. ①数据库—技术 IV. ①TP311.13

中国版本图书馆CIP数据核字(2014)第021514号

策划编辑: 张月萍

责任编辑: 贾 莉

印 刷: 北京中新伟业印刷有限公司

装 订: 三河市良远印务有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路173信箱

邮编: 100036

开 本: 787×1092 1/16

印张: 30.25

字数: 832千字

版 次: 2014年3月第1版

印 次: 2016年8月第4次印刷

定 价: 89.00元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888, 88258888

质量投诉请发邮件至zltz@phei.com.cn, 盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式: 010-51260888-819 faq@phei.com.cn。

用户体会

报表管理是销售管理的一项重要工作，面对70多个国家合作伙伴的不同种类型的销售报表，如何通过自动化手段进行格式统一、计算和分发，是我2009年在新兴市场集团工作时的“痛点”——是我特别想实现的，而这套方案必须要基于开源的方案来控制开发及未来维护的成本。

Jason给我推荐开源的ETL工具Kettle来完成这个工作，他使用 Kettle 加上开源的报表工具 Jasper，用了两个月的时间就把这套系统实现并上线了：

它可以自动从各个数据源获取数据、自动生成 Excel 报表，并自动投送到相关业务人员的邮箱里，这节省了我三个做报表的人力！而且数据更及时、准确！非常了不起！！

赵海生

客户数据与市场秩序 总监

联想集团

数据是投资的重要基础，但由于数据量大且指标较多，从各种不同格式的报告中摘取我们希望的数据一直是让我们头疼的事情。这一事情的改观发生在2011年Jason为我们带来Kettle工具之后，经过几个月的开发和测试，我们的指标自动抓取系统正式上线并一直沿用至今，它能从各种格式的报告中摘取重要的数据，这些数据形成我们分析的基础。实际上，这只是使用了Kettle工具的一小部分功能而已，相信在数据抓取和处理领域，我们还将有更多的合作机会。

从2011年就听到Jason要翻译本书的计划，很高兴能看到这一目标最终实现，这是Jason本人的一个里程碑，也是让更多的人受益于Kettle工具的一次契机，祝Jason和Kettle的路都越走越宽。

郑伟征

业务总监

北京中能兴业投资咨询有限公司

从企业架构的角度来看，和传统的编写代码相比，ETL工具在开发实施效率（包括代码复用）、可靠性、低出错率、可维护性上绝对都是巨大的进步。我个人相信在企业ETL领域，编程语言的工作未来可能会减少到总任务量的10%，剩下90%均需要借助ETL工具来实现。

本书和市面上林林总总的介绍ETL工具的书籍不同。书籍的原作者Matt是Kettle的核心设计与开发者（Kettle的灵魂所在），而且Jason带领的团队对 Kettle的源代码有深入的了解，并有丰富的实践经验，他们对本书所涉及的主题有切身的体会，这样可以最大限度地避免出现很多计算机译本图书出现的读者“不知所云”的情况。本书亦可以看作是Jason在国内不遗余力推广Kettle ETL解决方案的又一个里程碑。

徐洋

葛兰素史克（中国）投资有限公司 Enterprise Architect

数据对于每个企业来说都是极其重要的，它蕴含的价值不可限量，尤其是对于我服务的电信行业。我们做ETL工作先是自己开发程序或者存储过程，后来慢慢转型使用ETL工具，深刻体会到工具对于生产力发展的重要性。

随着对ETL的认识越来越深入，要求也越来越高，尤其是一些企业特定的需求即使是顶级的商业ETL软件也无法满足。在2012年一次偶然的机会我认识了Jason从而认识了Kettle，在Jason的帮助下仅用了三天时间就开发出一个定制化的组件，解决了长期困扰我们的问题，Kettle本身的灵活性、扩展性再加上Jason的团队对这款软件的驾驭能力，都是我们公司所需要的。

在得知Jason要翻译此书时，真的非常期待并衷心祝愿Jason和Kettle在中国能有更好的发展。

黄磊

上海理想信息产业（集团）有限公司 BI架构师

译者序

对于Kettle这个软件，要从2006年年初说起，当时我所在的公司内有很多数据整合类项目，当时商业的ETL工具太昂贵，不适合在预算有限的项目内使用，而手工写代码或写存储过程又会耗费大量的开发时间，还会增加项目的维护成本，于是我计划开发一款ETL工具，来满足公司内部数据整合项目的需求。作为产品开发的第一步，就是要调研当时市场上同类的开源产品，我调查了包括Kettle、CloverETL、OctopusETL、KETL等在内的多款开源ETL工具。

当时Kettle 2.2版本刚发布，在阅读完Kettle 2.2的代码后，我觉得这就是我想要的ETL工具，它的插件丰富功能强大，并行执行效率高，最重要的是它有灵活可扩展的插件架构，可以通过二次开发来提供更多的功能，并可以作为引擎方便地集成在第三方应用中。

于是我放弃了自己开发一个ETL工具的想法，转而成为一个Kettle开发人员。当时的Kettle刚刚问世，功能还有很多不完善的地方，于是我经常在MSN上和Kettle的作者Matt Casters先生讨论Kettle的一些技术细节。Matt是一位非常热心的程序员，对于我的问题或建议都能给予耐心的解答。这里要感谢Matt，正是因为他的无私奉献，才使Kettle成为世界上最流行的开源ETL工具。

2010年Pentaho Kettle Solutions一书出版，我通过朋友李小凡获得了该书，李小凡是www.itisbi.com网站的站长，他也为Pentaho在中国的推广做出了重要贡献。

看过该书后，我觉得这是一本非常好的介绍Kettle的书籍（尽管之前还有其他两本介绍Kettle的书籍，但那两本书基本是Kettle的入门书籍），Pentaho Kettle Solutions一书内容全面，对Kettle的技术细节也有一定程度的阐述，具有一定的理论深度。于是我计划翻译该书，以便让更多的做数据整合工作的开发人员都了解这个开源的ETL工具。

尽管从2010年年底我就开始动手翻译本书，但其间因为各种项目的耽搁，本书的翻译工作断断续续持续了三年。在翻译本书的过程中我还得到了www.pentahochina.com社区里一些朋友的支持，包括阿虎、银杏树、Super-超，他们协助我翻译了本书的一部分内容，在此也表示感谢。在2013年年中，本书即将翻译完成的时候，遇到中山大学的黄志洪教授，黄教授推荐了电子工业出版社出版该书。这里感谢黄志洪老师、张月萍老师的帮助。

在本书的翻译过程中，尤其要感谢父母和妻子对我的支持，我以翻译本书为由逃避了大部分的家务劳动。我的妻子曹雪梅除了承担大部分的家务劳动外，还承担了本书的一部分翻译工作。

在本书的成书过程中，我还得到了很多朋友和同事的帮助，包括曾浩、刘小鹏、郭振未、方进、任潇楠、郑发林、贺警阳、李超、冯海军、孙斌、饶丽、刘赫扬等同事，他们都为本书的出版做出了贡献，在此一并表示感谢。

现在我已经从一个 Kettle 开发人员变成了 Pentaho/Kettle 的粉丝，并且成立了一家商业智能/数据整合类的咨询公司（北京傲飞商智软件有限公司），从事 Pentaho/Kettle 的咨询、培训、开发、实施等工作（公司是 Pentaho 的官方合作伙伴，是 Pentaho 公司在中国地区唯一授权培训合作伙伴）。

由于本书是基于Kettle 4.0版本的，而Hadoop等大数据技术在Kettle 4.3版本中才支持，所以本书没有介绍Kettle的大数据功能。关心Kettle如何支持大数据的读者可以到Pentaho的wiki去了解。

虽然译者已经尽力去翻译本书，但出于译者能力所限，如在书中出现错误和疏漏，还请读者不吝指正。

关于作者

Matt Casters是一位具有多年工作经验的独立商业智能顾问。他为许多大公司建立了无数个数据仓库和BI解决方案。在过去的8年里，Matt Casters把自己的时间都用于研发一个ETL工具——Kettle。2005年12月，Kettle成为开源软件。2006年初期，Kettle走进Pentaho。随后，Matt就职于Pentaho，成为数据集成总监。在Pentaho，他继续从事Kettle的研发工作。Matt致力于帮助建设Kettle社区，回答论坛上的提问，有时在世界会议上发表演讲。博客：<http://www.ibridge.be>。Twitter: @mattcasters。

Roland Bouman目前从事前台页面和商业智能的研发工作。他从1998年开始从事IT行业。多年来一直致力于开源软件的研发，尤其是数据库技术、商业智能以及页面开发框架。同时，Roland Bouman还是MySQL和Pentaho社区的成员。他经常参加MySQL使用者会议、OSCON、Pentaho社区等国际会议。Roland Bouman不仅是MySQL 5.1 Cluster Certification Guide和Pentaho Solutions两本书的合著者之一，也是MySQL和Pentaho相关书籍的技术评论家。技术博客：<http://rpbouman.blogspot.com>。Twitter: @rolandbouman。

Jos van Dongen是一位著名的商业智能专家、作家和演说家。他从1991年开始从事软件开发、商业智能以及数据仓库等领域的工作。Jos van Dongen曾先后就职于顶级的系统集成公司和管理咨询公司。1998，他创立了自己的咨询公司，Tholis Consulting。他为许多商业和福利组织构建了BI和数据仓库系统。Jos为丹麦Database Magazine撰写了新的BI研发成果，并且经常在国内和国际会议上发言。Jos van Dongen撰写了一本关于开源BI的书，并且和Roland Bouman合作编写了Pentaho Solutions。更多信息参考：<http://www.tholis.com>。Twitter: @josvandongen。

致谢

按照惯例, 本书的封面列出了参与这本书创作的人员名单。但实际上, 这本书是许多人共同劳动的成果。正是这许许多多人在幕后辛勤地工作着, 才成就了本书的出版。作为作者, 我们真心地感谢所有为本书做出贡献的人们。感谢大家帮助我们完成了 *Pentaho Kettle Solutions*。

首先感谢为本书直接做出贡献的作者们。

- Ingo Klose提出了在单个转换中生成偏移键的解决方案。(参考第8章“处理维度表”的“基于计算器生成代理键”一节, 图8-2。)
- Samatar Hassan提供了Kettle支持RSS的转换例子。在第21章“Web Services”中, RSS部分几乎都是Samatar Hassan的贡献。
- Mike Hillyer和MySQL团队创建并一直维护着Sakila样例数据库。他们的Sakila样例数据库会在第4章有详尽阐述, 本书的其他章节也会引用Sakila样例数据库。
- Kasper de Graaf是本书的第四位作者, 虽然书的封面只出现了三位作者。他来自DIKW-Academy, 为本书写了Data Vault一章, 阐述了自己对这个领域的专业见解。Johannes van den Bosch审阅了Kasper的工作, 让Data Vault这一章更加清晰完整。
- Bernd Aschauer和Robert Wintner均来自Aschauer EDV (<http://www.aschauer-edv.at/en>), 两人为第6章SAP部分提供了例子和屏幕截图。
- Daniel Einspanjer来自Mozilla Foundation, 为第7章提供了转换例子。

感谢你们的辛勤工作, 为本书做出了巨大的贡献。同时感谢所有Pentaho和Kettle的技术专家, 他们在繁忙的工作日程中抽出时间审阅此书, 使得此书的羽翼丰满。

我们还要感谢Kettle和Pentaho社区。在整个写书前后, 社区中的伙伴们对ETL、数据整合和Kettle提出了宝贵的意见和建议。我们希望这本书对于正在使用和计划使用Kettle的朋友们起到实际的作用。读者是判定我们是否成功的最好证明。如果成功了, 我们要感谢Kettle和Pentaho社区中的每一位伙伴。

感谢Kettle软件项目的所有贡献者和研发人员。我们大家都是Kettle的热心使用者。我们热爱Kettle, 因为它用直接有效的方法解决了日常中的数据整合问题。使用Kettle工作快乐无比, 这也是完成这本书的巨大动力。

最后, 感谢Wiley出版社。感谢他给我们提供了写这本书的机会, 感谢他完美的管理体系和对我们的巨大支持。我们还要尤其感谢项目主编Sara Shlaer。尽管我们总是延期交稿, 但是Sara一直耐心鼓励着我们, 她的建议、关心、镇静和无与伦比的幽默感让本书及时地在截止日期前完成。另外, 还要感谢执行编辑Robert Elliot, 感谢他对我们小小团队的信任以及他对Pentaho

Kettle Solutions所做的努力。

——作者的话

由于当时正忙于发布Kettle 4，日程安排非常紧，所以单靠一个人的力量写这本书是极其困难的。感谢Jos和Roland，感谢他们的认真和专业，我们一起努力完成了这本书。也同时感谢他们接受了我的邀请。即使写书的过程很痛苦，但是和Jos、Roland一起合作，这一切都是值得的。

在Kettle的源代码还没有公开之前，Kettle还不太为人所知，因为没有太多的人会关注一个不开源的ETL工具。Kettle从当时的无人所知到现在成为世界上最广泛使用的开源ETL工具，这要感谢成千上万的志愿者们，他们帮助解决了很多问题。自从Kettle开源以来，Kettle和Kettle社区都快速发展。Kettle社区的贡献功不可没。正是由于这些研发人员、翻译人员、测试人员、错误报告人员、论坛参与人员、那些带来新想法的人们，甚至那些爱抱怨的人们，造就了今天的Kettle。这里我尤其要感谢我们社区中一个很重要的成员，Pentaho。Pentaho CEO Richard Daley和他的团队自从加入进来，一直都在支持着Kettle项目。没有他们的支持，Kettle不会取得今天的成绩。对于我来说，能和Pentaho的伙伴们一起工作是我的荣耀。

我们社区的几位成员对本书的部分技术内容进行了审阅。Nicholas Goodman、Daniel Einspanjer、Bryan Senseman、Jens Bleuel、Samatar Hassan以及Mark Hall审阅了我写的章节。他们提出了中肯的意见和建议。这是我第一次写书，难免有不妥之处，感谢他们花费宝贵时间和辛勤劳动。

——Matt Casters

首先我要感谢和我一起写这本书的两位合著者Jos和Matt。和这样经验丰富、知识渊博的专家们合作，是我的荣幸。希望今后有机会我们能够继续合作。我们彼此不同的技术领域帮助完成了这本书的不同方面。

此外，我还要感谢对我写的章节进行审阅的专家们，Benjamin Kallman、Bryan Senseman、Daniel Einspanjer、Sven Boden和Samatar Hassan。他们的评论、建议、直率中肯的批评促使书的写作更加完善。

最后感谢我博客<http://rpbouman.blogspot.com/>上的读者们。他们的评论给了我极大的鼓舞。当我公布要写Pentaho Kettle Solutions后，收到了许多好的反馈。

——Roland Bouman

回顾2009年10月，那时Pentaho Solutions才刚刚出版两个月。Roland和我一致同意不会再写另外一本书。但Bob Elliot找到了我们，建议我们再写一本。的确，我们也一直在讨论这个问题，一直认为如果再写一本书的话，那一定是关于Kettle的书。这也正是Bob想让我们做的，写一本用Kettle解决数据整合问题的书。接着我们很快发现Matt Casters也要成为书的作者，并向我们发出了一同写书的邀请。我们欣然接受。回顾过去，我不敢相信我们取得了这么大的成功。真心感谢Roland和Matt一直以来容忍我，接受我。感谢Bob。尤其感谢Sara，感谢她不懈的努力让我们圆满完成任务。

尤其要对Ralph Kimball说一声谢谢。读者会在书中找到他的观点。Ralph允许我们使用Kimball Group的34种ETL子系统作为这本书许多内容的框架。Ralph也审阅了第5章，他为第5章写出了极其丰富的评论，使得第5章为书中第二、三、四部分奠定了坚实的基础。

最后感谢Daniel Einspanjer、Bryan Senseman、Jens Bleuel、Sven Boden、Samatar Hassan以及Benjamin Kallmann，他们对我写的章节给予了技术的审阅，他们的评论、疑问和建议增添了书的价值。

——Jos van Dongen

介绍

50多年以前，以普通应用为主的计算机首次出现，计算机的应用进而延伸到科学和商业领域里。在过去，大多数公司基本只有一台计算机，连接一台显示器和一台打印机。因此，想要整合存储在不同系统中的信息是不可能的。然而，19世纪70年代后期，关系数据库的出现改变了这一切。80年代，计算机和数据库对公司信息的采集进一步升温。毫无疑问，这一切推动了一个新行业的产生，正如IBM研究者Dr. Barry Devlin 和Paul Murphy的会议论文所说：“一个商业和信息系统的建筑”（1988年 IBM Systems Journal首次出版，27卷，1号）。作为“商业报表的单一逻辑数据库”，商业数据库的概念首次出现。随后不到五年的时间，Bill Inmon 出版了*Building the Data Warehouse*，这本书具有划时代的意义，进一步推动了构建逻辑数据库的概念和科技。

数据仓库领域的一个重要概念就是数据整合。数据整合就是把不同数据库中的数据组合到一起，对外提供统一的数据视图。数据整合最典型的案例就是整合存货数据和订单数据。数据整合的另一个案例就是把各个部门的客户关系管理系统中的客户信息整合到公司客户关系管理系统中。

说明：阅读本书，你会看到有两个术语“数据整合（data integration）”和ETL（extract, transform, and load）在交叉使用。尽管技术上并不完全正确（ETL只是数据整合的一种，见第1章），但大多数开发人员都把这两个词视为同义词，很多年来我们也一直如此。

在理想情况下，不应该存在数据整合的需求。企业运营所需要的所有数据都应该保存在一个单一的系统里，所有的主数据都应该100%正确，所有分析和决策使用的外部数据都应该自动链接到我们的数据上。这个系统可以存储所有的历史数据，也可以以秒级的速度查询和分析数据。

遗憾的是，我们并非生活在理想世界中。在现实世界中，大多数的组织为各种目的使用不同的系统。有CRM系统，有会计系统，有销售和销售支持系统，有物流系统，有库存管理系统，等等，这个列表还在不断增长。更糟糕的是，业务上相同的数据可能会保存在不同的系统中，但内容并不完全相同。

为了能够处理所有这些问题，需要创建一个单一的、合成的、一致并且可靠的数据库来展现分析数据，由此看来，数据整合工具是必不可少的。目前最流行、功能最强大的数据整合工具是Kettle，也被称为Pentaho Data Integration。这正是本书讨论的主题。

Kettle的起源

Kettle起源于十年以前，本世纪初。当时，ETL工具千姿百态，比较流行的工具有50个左右，ETL框架数量比工具还要多些。根据这些工具的各自起源和功能可分为以下4种类型，如图1所示。

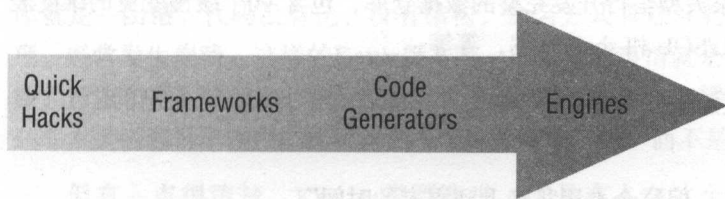


图1 ETL工具

- **快速代码修改 (Quick Hacks)**：这类工具主要用于抽取数据和加载文本文件。很多这类工具现在仍在使用。“hacker”和“hacking”这样的词汇现在成了贬义词。商业智能本身比较复杂，在很多情况下quick hacks是项目成功与否的关键，而且能够节省时间和成本。这种quick hacks的解决方案主要由咨询公司创造，一般是跟随项目的一次性解决方案。
- **框架 (Frameworks)**：通常情况下，当一个商业智能顾问同时做几个相似的项目时，代码可以在小范围内调整就可以应用到不同的项目上。这样说来，每个咨询公司都有自己的framework，因为这些frameworks帮助构建了ETL程序。而且通过改变参数，就可以完成不同项目的抽取数据、加载、日志、信息捕获、数据库连接等工作。
- **代码生成器 (Code Generators)**：当frameworks上再加一个开发界面作为额外的抽象层时，就可以基于元数据为某个平台（C、Java、SQL，等等）生成代码了。这种代码生成器的种类很多，有的是生成简单代码，还需要你手工维护代码，有的功能强大的ETL工具可以生成各种代码。这类ETL工具一般也是由一些比较著名的咨询公司开发的。
- **引擎 (Engines)**：随着ETL技术的不断发展，ETL引擎技术出现了，这样就不必再生成代码。引擎可以执行参数化的可配置的ETL过程，也就是对ETL本身的描述。这样就避免了生成代码、编译、部署等困难。

从大概的统计来看，超过一半的项目使用的是快速代码修改和框架的方法。剩下的项目里大部分使用各类基于代码生成器的ETL方法。基于引擎的ETL方法只在少量的ETL项目里使用，主要在非常大的ETL项目里使用。

说明：十年前，开源的ETL项目非常少。Enhydra Octopus是其中的一个，它是一个基于Java的，代码生成器类型的ETL工具（可以从www.enhydra.org/tech/octopus下载）。很多用户从这个开源项目受益，所以这个项目现在仍旧可以使用，它也是开源项目可以持续不衰的一个例子。

正是在这样的背景下，Kettle软件的作者，本书作者之一，Matt Caster每天忙于咨询工作，为不同项目不停地修改ETL代码和框架，部署各种ETL工具的代码生成器。

时间回退到2000年，Matt还是商业智能顾问，他通常的职位是数据仓库架构师或管理

员。在这样的职位上,经常需要把企业数据转换为业务需要的各种信息。通常这些工作是在没有一个很强大的ETL工具的情况下完成的,因为这类的ETL工具都很昂贵。尤其对中小型项目来说,不可能使用这么昂贵的工具。在这种情况下,没有什么选择:一次次面对同样的问题,使用各类框架或代码生成技术来完成自己的工作。他做过相关的工作包括:用C语言和嵌入式SQL(ESQL/C)从Informix里抽取数据;用Microsoft VB开发一个抽取工具,从IBM AS/400 Mainframe系统里抽取数据;为一家大型银行开发完整的数据仓库,包含90个缓慢变更的维度表和35个事实表;完全手写Oracle PL/SQL和shell脚本,等等。

但是这些经历,使Matt明白应该做些什么。在2001年Matt就有了开发一个自己的ETL工具的想法。

Matt: “我想写一个ETL软件。但这会占用我晚上和周末的时间”

Kathleen (Matt的夫人): “Oh,太好了!要用多长时间?”

Matt: “如果一切顺利,第一个能运行的版本应该用三年时间,第一个完整的版本要用五年。”

Kettle 的设计

因为十年来一直在同各种ETL工具做斗争,所以Matt确定了Kettle的一个主要设计目标是尽可能开放。主要就是指:

- 开放,可读的元数据格式(XML)。
- 开放,可读的关系型资源库格式。
- 开放的API。
- 容易安装(少于2分钟)。
- 对各类数据库开放。
- 容易使用的图形用户界面。
- 容易传送数据。
- 容易把数据转换成各种格式。

在最开始的两年,进度比较缓慢,大量的工作都用于研究这个新的ETL工具应该具有哪些功能。开发一个并行化的ETL引擎就是在这段时间确定的。多年的基于代码修改方式、框架方式、代码生成器方式的ETL项目惨痛经验,使Matt确信,新的ETL工具一定是应该基于引擎方式的。

因为Matt以前主要使用C语言进行开发,所以开始时,他使用客户端/服务器这样的代码,来测试在不同的处理器和服务器之间传递数据的性能。通过很多不同场景的测试,他明白ETL性能瓶颈应该主要在于数据的编码和解码。所以,Kettle的一个设计原则就是尽量不做数据的转换。如今这一原则仍在Kettle中可以体现出来。

说明: Kettle一词起源于“KDE ETL Environment”,因为最开始的计划是在K Desktop Environment (www.kde.org)上开发这个软件。在这个计划被取消后,才把它重命名为“Kettle ETL Environment”。

由于缺少各种关系数据库的驱动，最终还是采用了当时较新的Java开发语言。在2003年，又选择了SWT (Standard Widget Toolkit) 来开发界面，因为Matt不喜欢Java AWT (Abstract Window Toolkit) 的性能和界面风格。而SWT使用了本地操作系统的组件库，因此性能更好，而且界面也更符合操作系统的风格。

因为是Java新手，又要开发这个复杂的ETL工具，可以想象在开始的第一年，Kettle的代码库就是一团糟。代码没有包；没有结构，命名方式非常可笑（C语言的风格）。没有做异常处理，经常发生崩溃。这样的Kettle版本唯一能做到的事情就是它可以工作了。它可以读取文本文件，读取和写入数据库，而且它有了一个JavaScript 脚本步骤，用来解决各种复杂问题。而且它还非常灵活和易于使用。这毕竟是一个商业智能工具，而不是一个Java项目。

但有一点很清楚，Kettle 需要改进。于是有人在这个时候提供了帮助，朋友 Wim De Clercq，他是ixor (www.ixor.be) 的合伙人，也是一位高级Java 架构师。他解释了很多Java 概念，如包、异常处理等。慢慢地时间就在学习Java设计模式（如单例模式等）中流走了。

听朋友的建议就意味着大量的代码重构。所以第一个版本后，Matt 把每个周末的时间都用到了代码重构上，重新写了几万行代码。逐渐的，几个月以后，事情就开始向着好的方向发展。

Kettle 获得机会

Matt和朋友、同事、其他高级商业智能（BI）顾问们分享了Kettle的最初成就，听取了他们对Kettle 的想法，从那时开始 Kettle 开始逐渐被大家了解。后来在2004年，Matt 把 Kettle 部署在了比利时Flemish 交通中心 (www.verkeerscentrum.be)，Kettle从遍布比利时的几千个数据源整合几亿条数据。当时项目小组没有时间写代码，也没有钱去买那些商业ETL工具，所以项目就使用了Kettle。这个比利时交通中心的项目提升了Kettle 的功能和性能，在这个时期，Kettle 发展非常迅速。因为有了各种数据的测试用例，Kettle可以更好地支持各种数据库。也是在这个时期，世界各地的开发人员都可以免费下载Kettle版本使用（免费下载，还没有开源）。

最开始反馈不多，但大多数的反馈都是积极的。最有趣的一个反馈来自于一个德国的开发人员，Jens Bleuel，他问是否可以把第三方的软件整合进Kettle，他希望把SAP/R3 connector整合到Kettle。Kettle 当时的版本是1.2，还没有插件架构。Jens Bleuel的这个需求是当时开发插件框架的主要原因，后来就形成了 Kettle 2.0。最后直到2004年底，才开发完成。这是一个相对完整的版本，支持缓慢变更维度、杂项维度、28个步骤和13种数据库。直到这个时候，这个工具的真正潜力才发挥出来。然后，Jens Bleuel 创建了Kettle 的第一个插件，ProSAPCON，用来从SAP/R3 服务器读取数据。

Kettle 走向开源

在那个时期有很多令人激动的事情，Matt 和 Jens 打算把Kettle商业化，通过kettle.be 网站和Jens所在的Proratio (www.proratio.de) 公司来销售Kettle。

Kettle 还是在取得进展，也有一些试用的请求。但是，做一个完整的ETL工具的开发和销售是一项艰巨的任务，不是靠个人可以完成的。而且Matt还发现，用 Kettle来工作很有乐趣，而销

售Kettle 却没什么乐趣。他不得不找到一种方式，可以把精力聚集到有乐趣的开发工作中。所以最后在2005年的夏天，Matt决定把Kettle 开源。这样Kettle可以自己卖自己，并吸引更多的人参与开发工作。

2005年12月份，Kettle 2.2 的代码第一次发布后，反响非常强烈。JavaForge 上第一周的下载数量就达到了35 000 次，消息迅速在全世界传播。

因为很多开源项目到最后都成了无人管的项目，为了避免这种情况的发生，要尽快为Kettle项目构建一个社区。这就意味着，在随后的几年可能需要回答上千封的电子邮件和论坛帖子。幸运的是，Kettle很快获得了开源商业智能公司Pentaho的帮助（www.pentaho.com），Pentaho获得了源代码的版权，Matt也成为 Pentaho的内部人员，带领Kettle项目的开发，随后Kettle改名为Pentaho Data Integration。

Kettle同样获得了来自世界各地的开发人员、翻译人员、测试人员、文档编写人员的帮助，没有这些人的帮助，Kettle 也不能像今天这样发展迅速。

关于本书

本书起源于2009年8月，此时Roland 和 Jos 的第一本书《Pentaho 解决方案》（即Pentaho Solutions）已经出版。正如跑过马拉松的人，他们发誓以后“再也不跑了”。但是当看到第一本的热烈反响，他们的态度慢慢地转变过来了。他们想如果还要写一本书的话，就要写Kettle和数据整合。当《Pentaho 解决方案》一书的出版人Bob Elliot问他们能否再写一本Kettle书的时候，Kettle书的主题和目录其实都已经确定下来了。另外，还让他们受到鼓舞（和吃惊）的是，他们的配偶都鼓励他们继续写作。还有其他的好消息，当他们邀请Matt Casters为本书审核时，Matt Casters也希望参与到本书的写作中，这当然是求之不得的事情了。

当时写《Pentaho 解决方案》一书的动力和主要原因，如今仍然存在：企业开始逐渐了解商业智能能给企业带来什么价值，并逐渐认识了开源和免费商业智能的解决方案。这些商业智能解决方案要求事先把数据集中在一起，然后再进行分析、报表和仪表盘等工作。所以商业智能项目的开始阶段都要先进行数据的整合，本书就可以帮助你进行数据整合的工作。

在过去的十多年里，各种类型的开源软件逐渐被大家认识和接受，来代替那些价格昂贵而不灵活的商业软件。人们常会错误地认为开源软件就是没有成本的，尽管从软件License的角度看这是正确的，但一个商业智能解决方案（永远）不会免费。在一个解决方案里有硬件成本、方案实现的成本、维护的成本、培训和移植的成本，等等，所有这些费用加一起，软件License的费用只占其中一小部分。开源降低了软件本身的成本，而且任何人都可以获得源代码，都可以发现源代码的bug，这样也提高了代码的质量。开源软件一般都是基于开放的标准和通用的编程语言（大部分是Java），这样使开源软件更灵活、更具扩展性。而且大多数开源软件都不局限在某个操作系统平台上，这样更扩展了软件的灵活性和自由度。

但开源软件缺少文档和手册。很多开源软件提供了高质量的代码，但开发人员因为更多地关注于软件本身而没有太多的时间写文档。尽管你可以找到很多关于Kettle的零散的信息来源，但我们觉得有必要为正在探索Kettle并构建数据整合解决方案的ETL开发人员提供一个完整的信

息来源。这就是本书的目的——帮助你使用Kettle 构建数据整合的解决方案。

谁需要读这本书

本书适合于想知道如何用Kettle来构建 ETL 解决方案的人。例如寻找低成本的ETL方案的IT经理、想扩充自己知识和技能的IT 专家、负责开发ETL方案的BI或数据仓库顾问。可能你是一个有丰富经验的软件开发人员，但对数据整合领域不太了解；也可能你是一个经验丰富的使用某些商业软件的 ETL开发人员，但对于Kettle不太了解。无论哪种情况，因为有了这本参考书，你就可以直接上手了。当然读者如果能有商业智能、数据库等方面的知识更好，但在本书的开始部分会介绍一些这方面的知识。当然本书也会介绍数据整合方面的概念，但重点还是讲解如何把这些概念转换为实际的工作方案。这也是本书被命名为《Pentaho Kettle 解决方案》的原因。

阅读本书的前提

只要能做到下面两件事，你就可以开始阅读本书了：有一台计算机，能连接到互联网。本书使用到的所有软件都可以从互联网下载。对计算机的系统也没有太多的要求，有1GB内存和2GB空闲硬盘的，一般在4年之内的计算机都可以很好地运行Kettle作业。

本书一些章节里有提供软件下载的URL地址。对于Pentaho软件，除了源代码，还提供有4种类型的软件版本：

- GA (General Availability) releases: 这是稳定的发布版本，并不是最新的版本，但是最可靠的版本。
- Release candidates: 候选版本，可能会成为下一个发布版本的版本，里面可能还会有一些未发现的bug。
- Milestone releases: 最新的里程碑版本，里面都会有一些新功能。
- Nightly builds: 每天的build版本，最新的版本，也是最不稳定的版本。

在写作本书的时候，我们使用的都是每天的build版本。在写作本书时，Kettle 4.0的GA版本已经发布了。也就是说，在你读本书的时候就可以使用没有多少bug的正式发布的稳定版本。

Kettle社区版的下载地址是：<http://wiki.pentaho.com/display/COM/Community+Edition+Downloads>。

从本书可以学到什么

本书会教给你：

- 数据整合是什么，数据整合的价值。
- Kettle 解决方案的概念基础。
- 如何在单机和客户/服务器环境下安装和配置Kettle。
- 如何使用MySQL的Sakila演示数据库构建一个完整的端到端的ETL解决方案。

- 34种ETL子系统，如何使用Kettle 实现这34种子系统。
- Kettle 如何完成数据抽取、清洗和确认、处理维度表、加载事实表、操作OLAP 立方体。
- Kettle 的开发生命期。
- 如何在Kettle 环境里使用Pentaho 的敏捷BI 工具。
- 如何调度和监控作业和转换。
- 多个开发人员如何协同工作，如何管理不同版本的ETL方案。
- 什么是数据的血统分析、影响分析、审计。Kettle 如何支持这些概念。
- 如何利用分区、并行化和动态集群技术提高Kettle 的性能。
- 如何使用复杂文件、Web 服务和Web API。
- 如何使用Kettle 加载基于Data Vault 模型的企业数据仓库。
- 如何在其他应用中集成Kettle，如何通过二次开发扩展Kettle。

本书如何组织

本书解释了ETL的概念、技术和解决方案。本书的很多场景都使用了MySQL的Sakila数据库，除此之外，我们还通过其他不同的场景来演示不同的概念。如果例子还依赖于其他数据库，我们都确保这些例子和MySQL数据库（5.1版本）兼容。

这些例子都提供了在实际环境中应用所必需的技术细节。例子范围覆盖了从部门级的数据集市到企业级的数据仓库。

第一部分：开始

本书的第一部分在一个较高的层次上，对Kettle 的架构、功能等做了一个快速的介绍。这一部分包括下面几个章节。

第1章：ETL入门。介绍数据整合项目的主要概念和挑战。我们介绍事务系统和分析系统的主要区别、ETL适用于哪些场合、如何用ETL解决方案的不同部分来解决数据整合问题。

第2章：Kettle基本概念。介绍了Kettle的一些基本设计原则，以及Kettle 的软件组成。我们介绍了Kettle的基本概念，如作业、转换、步骤、步骤之间的连接等，以及它们之间的相互作用。另外本章还介绍了Kettle 如何和数据库交互、设置数据库特定的参数。本章还提供了Kettle 界面的一个快速教程。

第3章：安装和配置。介绍如何获得Kettle 软件，如何安装Kettle。Kettle包含哪些组件，它们之间的关系，如何利用它们构建ETL解决方案。最后，我们解释不同的配置选项、配置文件和配置文件的位置。

第4章：ETL示例解决方案——Sakila。基于MySQL Sakila数据库，介绍一个完整的ETL解决方案的例子。使用本章的星型模型，你可以了解如何处理缓慢变更维度以及如何加载事实表。另外本章的一个重要主题就是如何通过映射步骤重用转换。

第二部分：ETL

本书的第二部分介绍的是Ralph Kimball 博士和他的同事们提出的34种ETL子系统。这些ETL子系统分别在*The Kimball Group Reader* (Wiley, 2010) 和*The Data Warehouse Lifecycle Toolkit* (Wiley, 2008) 书中介绍过。这一部分包括下面的章节。

第5章：ETL子系统。介绍不同ETL子系统和它们的分类。四个类别分别是抽取、清洗和确认、发布、管理。在本章，我们还介绍了Kettle 如何支持这些子系统。本章是其他章节的基础，也是理解ETL解决方案架构的必读材料。

第6章：数据抽取。它是ETL子系统的第一类，包括了数据剖析、变更数据捕获和抽取系统。我们解释什么是数据剖析，为什么它应该是ETL项目的第一个环节。变更数据捕获(CDC)用来检测源系统中变化的数据，我们提供了几个方案来实现CDC。

第7章：清洗和校验。它是ETL子系统的第二类，也是真正进行ETL过程的地方。在大多数情况下，并非只是简单地从不同的源系统中读取数据，再发布这些数据。数据必须要一致，要删除冗余和重复的数据；多种数据编码模式要统一成一种编码模式。本章介绍了很多Kettle用于这类转换的步骤，如模糊查询步骤用来进行模糊匹配和排重。

第8章：处理维度表。这是第三类ETL子系统“发布”的一部分。在这一章，我们先介绍什么是维度表。接着介绍如何使用“维度查询/更新”步骤进行不同类型的加载和更新。然后再特别介绍子系统10，代理键生成系统。其他的主题，如时间维度、杂项维度、小维度等，在本章做了一些介绍。最后介绍了递归层次。

第9章：加载事实表。本章介绍加载不同类型的事实表，本章的前半部分介绍不同的加载策略，并介绍如何处理早到和迟到的事实数据。我们介绍并演示了不同的批量加载方法。除了最常见的交易事实表外，我们还介绍了周期和累积事实表。我们还介绍了一种新型的事实表，“面向状态”的事实表。

第10章：处理OLAP数据。本章全部用来介绍ETL子系统中的一个(20, OLAP cube 构建系统)。本章演示了如何处理三种类型的OLAP数据源：从XML/A 和Mondrian 的 cube 中读取数据，读取和加载Polo cube中的数据。

第三部分：管理和部署

本书前面部分介绍如何构建解决方案，本部分的章节主要介绍如何部署和管理解决方案。

第11章：ETL开发生命期。本章主要介绍如何使用Kettle中的工具，设计、开发和测试一个ETL 解决方案。本章介绍了敏捷BI开发方法，以及如何通过敏捷BI方法加快开发过程。本章介绍了不同的测试方法，在解决方案发布之前都要使用这些测试方法进行测试。

第12章：调度和监控。本章介绍了不同的调度方法。包括操作系统调度工具，如cron和Windows计划任务，以及Pentaho 内置的Pentaho BI调度平台。一般可以直接在设计环境中监控正在运行的作业和转换，但我们也介绍了如何通过日志表来获取成功或失败的作业信息。

第13章：版本和移植。本章介绍如何管理不同版本的 Kettle作业和转换，通过版本管理，可以在必要的情况下，把作业回滚到前一个版本。本章介绍的另一个主题是如何在开发、测试、确认、生产环境中完成 Kettle作业的移植。

第14章：血统和审计。本章介绍如何使用Kettle的元数据来判断数据的来源以及数据如何被使用。出于审计的目的，我们需要知道作业何时运行，运行了多久，给数据带来了哪些变化。为了完成这些需求，Kettle 有可扩展的日志能力，这些我们在本章都做了详细介绍。

第四部分：性能和扩展性

本部分内容都是关于Kettle的性能问题。Kettle里有一些选项可以加快抽取、转换和加载的速度，本部分里的每一章都是Kettle性能问题的一种解决方案。

第15章：性能调优。本章解释Kettle 转换引擎的工作原理，帮你找出Kettle 转换的性能瓶颈。我们也提供了关于如何提高Kettle性能和吞吐量的几种解决方案。本章的大部分篇幅都用于介绍如何提高文本文件的处理速度，以及如何把数据尽可能快地加载到Kettle的数据流里。

第16章：并行、集群和分区。本章介绍了关于提高Kettle 性能的更多技术内容。一般来说有两种方法：水平扩展和垂直扩展。垂直扩展的目标是充分利用单机的处理能力，充分利用多核CPU的大内存。水平扩展是把任务分布在不同的服务器上。我们介绍如何在Kettle 里使用这两种方法来提高性能。

第17章：云计算中的动态集群。本章介绍如何使用云计算这样的大规模计算能力来提高 Kettle 性能。如何把第16章介绍的集群方式应用到云计算的环境中，我们使用Amazon的 Elastic Computing Cloud (EC2) 的云计算环境。本章另外还介绍了如何基于不同的工作负载，来动态扩展或缩减计算集群，以节省成本并提高性能。

第18章：实时数据整合。本章介绍Kettle如何处理源源不断的实时数据流。Kettle本身就是基于流的数据处理引擎，所以只要把Kettle连接到一个持久的数据流或消息队列，并启动转换，Kettle就可以处理实时数据流了。

第五部分：高级主题

本书的高级主题覆盖了从几个方面演示了Kettle的一些高级功能、如何扩展Kettle 的能力、如何把Kettle用在第三方应用里。

第19章：Data Vault管理。本章解释了什么是数据仓库的Data Vault (DV) 模型，如何使用 Kettle 加载DV模型中的三类表：中心表、链接表、附属表。

第20章：处理复杂数据格式。本章介绍如何处理非关系型数据，包括结构化和半结构化的数据。介绍如何把键值对格式的数据转化为规则的数据，介绍如何使用正则表达式来结构化那些看上去是非结构化的数据，介绍如何处理重复分组和多值属性。这种键值对数据一

般是存在所谓的无模式或NoSQL数据库中的。

第21章：Web Services。本章介绍如何从互联网获取数据。介绍和Web 相关的一些组件，并描述了从互联网获取数据的不同方法，如HTTP GET、POST和SOAP。另外介绍了如何使用Kettle处理一些常见的互联网数据格式，如XML、JSON和RSS。

第22章：Kettle集成。本章演示了Kettle如何被外部应用使用。描述了Kettle API和几个例子。最简单的一个例子就是Pentaho 报表，它使用Kettle 转换作为数据源。

第23章：扩展Kettle。本章介绍如何写自己的插件来扩展Kettle。介绍了如何开发不同类型的插件：包括步骤、作业项、分区类型、资源库类型和数据库类型。

附录

本书附录部分提供了如下一些快速参考。

附录A：Kettle 生态群。描述了在Kettle的世界里都包括什么，如何获得帮助，如何使用论坛和其他用户沟通。我们还解释了如何使用Jira，Jira是Pentaho的问题跟踪管理系统，通过Jira我们可以查找和追踪Bug，监控Bug状态，并查看软件发展的路线图。

附录B：Kettle企业版特性。解释Kettle企业版和社区版的不同，详细说明了企业版的一些特性。

附录C：内置的变量和属性参考。提供Kettle内置的所有变量和属性的参考，在Kettle解决方案里可以直接使用这些变量和属性。

前提

本书主要介绍的是Kettle，Kettle的安装和使用是本书的基本内容。第3章描述了Kettle的安装和配置过程。但本书在介绍Kettle的过程中，还用到了其他一些软件，这些软件也需要我们先了解，这里统一做介绍。

Java

Kettle 运行在Java 平台上。尽管Java 无处不在，而且你的机器上可能已经存在Java，不过我们在第3章中还是提供了安装向导和注意事项。

MySQL

MySQL数据库是本书所有例子的默认数据库。可以从 <http://dev.mysql.com/downloads/mysql> 网站下载MySQL。MySQL数据库几乎可以安装在任何操作系统下，可以通过下载后运行Installer来安装，也可以通过Linux 的系统资源库来安装。如何你用的是Ubuntu系统，而且还没有安装MySQL数据库，可以通过运行`sudo apt-get install mysql-server` 命令来安装MySQL数据库。另外还

要下载MySQL的GUI客户端。从MySQL 5.2开始, MySQL Workbench 客户端提供数据库建模、管理和查询功能, 成为一个集成的解决方案。

SQL Power Architect

为了给数据库建模, 我们强烈推荐使用数据库建模工具, Kettle并不提供这个功能, Pentaho 其他工具也不提供这个功能。最好的一个开源的建模工具就是SQL Power Architect, 本书也使用了这个建模工具。可以从SQLPower 的网站下载这个工具: www.sqlpower.ca/page/architect。

Eclipse

Kettle使用Java开发, 使用了Eclipse IDE集成开发环境, 本书的最后一章介绍了如何使用Eclipse开发自己的Kettle插件。Eclipse是一个全能工具, 任何开发语言都可以使用Eclipse开发环境。因为Eclipse的框架灵活, 使用不同的插件和视图, 它还可以完成诸如数据建模、创建报表、数据挖掘之类的工作。Eclipse可以从www.eclipse.org/download 下载。如果你运行的是Ubuntu, 也可以从软件中心下载 (Ubuntu 10.4 开发工具→IDE, Ubuntu 9.10开发工具→Programming) 或运行命令 `sudo apt-get install eclipse`。

网络资源

本书用到的所有例子都可以从本书网站下载 (www.wiley.com/go/kettlesolutions)。例子按照章节目录组织, 例子中的文件包括:

- 例子数据库的 Power*Architect 数据模型。
- 所有的PDI作业和转换文件。
- 例子必要的SQL 脚本。

更多的资源

本书的很多章节都给出了进一步阅读的资料或网站, 这些资源更深入介绍了或延伸了本书讨论的一些主题。如果你对商业智能和数据仓库还不太了解 (或者你想跟踪最新的发展), 下面有一些资源可以参考:

- http://en.wikipedia.org/wiki/Business_intelligence
- <http://www.kimballgroup.com>
- <http://b-eye-network.com>
- <http://www.tdwi.org>

你也可以访问我们的网站, 那里有我们的联系方式, 如果有需要你可以直接和我们联系:

- Matt Casters: www.ibridge.be
- Roland Bouman: rpbouman.blogspot.com
- Jos van Dongen: www.tholis.com

目录

第一部分：开始

| | |
|-------------------------|----|
| 第1章 ETL入门..... | 2 |
| 1.1 OLTP和数据仓库对比..... | 2 |
| 1.2 ETL是什么..... | 3 |
| 1.2.1 ETL解决方案的演化过程..... | 4 |
| 1.2.2 ETL基本构成..... | 5 |
| 1.3 ETL、ELT和EII..... | 6 |
| 1.3.1 ELT..... | 6 |
| 1.3.2 EII：虚拟数据整合..... | 7 |
| 1.4 数据整合面临的挑战..... | 8 |
| 1.4.1 方法论：敏捷BI..... | 9 |
| 1.4.2 ETL设计..... | 10 |
| 1.4.3 获取数据..... | 10 |
| 1.4.4 数据质量..... | 12 |
| 1.5 ETL工具的功能..... | 13 |
| 1.5.1 连接..... | 13 |
| 1.5.2 平台独立..... | 14 |
| 1.5.3 数据规模..... | 14 |
| 1.5.4 设计灵活性..... | 14 |
| 1.5.5 复用性..... | 15 |
| 1.5.6 扩展性..... | 15 |
| 1.5.7 数据转换..... | 15 |
| 1.5.8 测试和调试..... | 16 |
| 1.5.9 血统和影响分析..... | 16 |
| 1.5.10 日志和审计..... | 16 |
| 1.6 小结..... | 17 |
| 第2章 Kettle基本概念..... | 18 |
| 2.1 设计原则..... | 18 |
| 2.2 Kettle设计模块..... | 19 |
| 2.2.1 转换..... | 19 |
| 2.2.2 作业..... | 23 |
| 2.2.3 转换或作业的元数据..... | 28 |
| 2.2.4 数据库连接..... | 28 |
| 2.2.5 工具..... | 31 |
| 2.2.6 资源库..... | 31 |
| 2.2.7 虚拟文件系统..... | 31 |
| 2.3 参数和变量..... | 32 |
| 2.3.1 定义变量..... | 32 |
| 2.3.2 命名参数..... | 33 |
| 2.3.3 使用变量..... | 33 |
| 2.4 可视化编程..... | 34 |
| 2.4.1 开始..... | 34 |

| | |
|---------------------------------|----|
| 2.4.2 创建新的步骤..... | 35 |
| 2.4.3 放在一起..... | 36 |
| 2.5 小结..... | 38 |
| 第3章 安装和配置..... | 39 |
| 3.1 Kettle软件概览..... | 39 |
| 3.1.1 集成开发环境：Spoon..... | 40 |
| 3.1.2 命令行启动：Kitchen和Pan..... | 42 |
| 3.1.3 作业服务器：Carte..... | 42 |
| 3.1.4 Encr.bat和encr.sh..... | 42 |
| 3.2 安装..... | 43 |
| 3.2.1 Java环境..... | 43 |
| 3.2.2 安装 Kettle..... | 43 |
| 3.3 配置..... | 46 |
| 3.3.1 配置文件和.kettle目录..... | 46 |
| 3.3.2 用于启动Kettle程序的shell脚本..... | 51 |
| 3.3.3 管理 JDBC 驱动..... | 52 |
| 3.4 小结..... | 53 |
| 第4章 ETL示例解决方案——Sakila..... | 54 |
| 4.1 Sakila..... | 54 |
| 4.1.1 sakila示例数据库..... | 55 |
| 4.1.2 租赁业务的星型模型..... | 57 |
| 4.2 预备知识和一些基础的Spoon技巧..... | 60 |
| 4.2.1 安装ETL解决方案..... | 60 |
| 4.2.2 Spoon使用..... | 60 |
| 4.3 ETL示例解决方案..... | 61 |
| 4.3.1 生成静态维度..... | 62 |
| 4.3.2 循环加载..... | 64 |
| 4.4 小结..... | 80 |

第二部分：ETL

| | |
|-----------------------|----|
| 第5章 ETL子系统..... | 82 |
| 5.1 34种子系统介绍..... | 82 |
| 5.1.1 抽取..... | 83 |
| 5.1.2 清洗和更正数据..... | 84 |
| 5.1.3 数据发布..... | 86 |
| 5.1.4 管理ETL环境..... | 89 |
| 5.2 小结..... | 91 |
| 第6章 数据抽取..... | 92 |
| 6.1 Kettle数据抽取概览..... | 92 |
| 6.1.1 文件抽取..... | 93 |
| 6.1.2 数据库抽取..... | 97 |

6.1.3 Web数据抽取..... 98

6.1.4 基于流的和实时的抽取..... 99

6.2 处理ERP和CRM系统..... 100

6.2.1 ERP 挑战..... 100

6.2.2 Kettle ERP插件..... 101

6.2.3 处理SAP数据..... 101

6.2.4 ERP和CDC 问题..... 104

6.3 数据剖析..... 105

6.4 CDC: 变更数据捕获..... 110

6.4.1 基于源数据的CDC..... 111

6.4.2 基于触发器的CDC..... 113

6.4.3 基于快照的CDC..... 113

6.4.4 基于日志的CDC..... 116

6.4.5 哪个CDC方案更适合你..... 117

6.5 发布数据..... 117

6.6 小结..... 118

第7章 清洗和校验..... 119

7.1 数据清洗..... 120

7.1.1 数据清洗步骤..... 121

7.1.2 使用参照表..... 123

7.1.3 数据校验..... 127

7.2 错误处理..... 130

7.2.1 处理过程错误..... 131

7.2.2 转换错误..... 132

7.2.3 处理数据(校验)错误..... 133

7.3 审计数据和过程质量..... 136

7.4 数据排重..... 137

7.4.1 去除完全重复的数据..... 137

7.4.2 不完全重复问题..... 138

7.4.3 设计排除重复记录的转换..... 139

7.5 脚本..... 142

7.5.1 公式..... 143

7.5.2 Java脚本..... 143

7.5.3 用户自定义Java表达式..... 144

7.5.4 正则表达式..... 145

7.6 小结..... 146

第8章 处理维度表..... 147

8.1 管理各种键..... 148

8.1.1 管理业务键..... 148

8.1.2 生成代理键..... 149

8.2 加载维度表..... 154

8.2.1 雪花维度表..... 154

8.2.2 星型维度表..... 159

8.3 缓慢变更维度..... 161

8.3.1 缓慢变更维类型..... 161

8.3.2 类型1的缓慢变更维..... 161

8.3.3 类型2的缓慢变更维..... 163

8.3.4 其他类型的缓慢变更维..... 167

8.4 更多维度..... 168

8.4.1 生成维(Generated Dimensions) .. 168

8.4.2 杂项维度(Junk Dimensions) 169

8.4.3 递归层次..... 170

8.5 小结..... 171

第9章 加载事实表..... 172

9.1 批量加载..... 173

9.1.1 STDIN和FIFO..... 173

9.1.2 Kettle批量加载..... 174

9.1.3 批量加载一般要考虑的问题..... 176

9.2 维度查询..... 176

9.2.1 维护参照完整性..... 176

9.2.2 代理键管道..... 177

9.2.3 迟到数据..... 179

9.3 处理事实表..... 182

9.3.1 周期快照和累积快照..... 182

9.3.2 面向状态的事实表..... 183

9.3.3 加载周期快照表..... 185

9.3.4 加载累积快照表..... 185

9.3.5 加载面向状态事实表..... 186

9.3.6 加载聚集表..... 186

9.4 小结..... 187

第10章 处理OLAP数据..... 188

10.1 OLAP的价值和挑战..... 189

10.1.1 OLAP 存储类型..... 190

10.1.2 OLAP在系统中的位置..... 191

10.1.3 Kettle OLAP选项..... 191

10.2 Mondrian..... 192

10.3 XML/A服务..... 194

10.4 Palo..... 197

10.4.1 建立Palo 连接..... 198

10.4.2 Palo 架构..... 199

10.4.3 读Palo数据..... 200

10.4.4 写Palo数据..... 202

10.5 小结..... 204

第三部分: 管理和部署

第11章 ETL开发生命期..... 206

11.1 解决方案设计..... 206

11.1.1 好习惯和坏习惯..... 206

11.1.2 ETL流设计..... 209

11.1.3 可重用性和可维护性..... 209

11.2 敏捷开发..... 210

11.3 测试和调试..... 214

11.3.1 测试活动..... 214

11.3.2 ETL测试..... 215

11.3.3 调试..... 218

11.4 解决方案文档化..... 220

11.4.1 为什么实际情况下文档很少..... 220

11.4.2 Kettle的文档功能..... 221

11.4.3 生成文档..... 222

11.5 小结..... 223

第12章 调度和监控..... 224

12.1 调度..... 224

12.1.1 操作系统级调度..... 225

12.1.2 使用Pentaho 内置的调度程序..... 228

12.2 监控..... 232

12.2.1 日志..... 232

| | |
|-----------------------------|------------|
| 12.2.2 邮件通知 | 234 |
| 12.3 小结 | 237 |
| 第13章 版本和移植 | 238 |
| 13.1 版本控制系统 | 238 |
| 13.1.1 基于文件的版本控制系统 | 239 |
| 13.1.2 内容管理系统 | 240 |
| 13.2 Kettle 元数据 | 240 |
| 13.2.1 Kettle XML 元数据 | 241 |
| 13.2.2 Kettle 资源库元数据 | 242 |
| 13.3 管理资源库 | 244 |
| 13.3.1 导出和导入资源库 | 244 |
| 13.3.2 资源库升级 | 245 |
| 13.4 版本移植系统 | 245 |
| 13.4.1 管理XML文件 | 245 |
| 13.4.2 管理资源库 | 246 |
| 13.4.3 解决方案参数化 | 246 |
| 13.5 小结 | 248 |
| 第14章 血统和审计 | 249 |
| 14.1 批量血统抽取 | 250 |
| 14.2 血统 | 251 |
| 14.2.1 血统信息 | 251 |
| 14.2.2 影响分析信息 | 252 |
| 14.3 日志和操作元数据 | 254 |
| 14.3.1 日志基础 | 254 |
| 14.3.2 日志架构 | 255 |
| 14.3.3 日志表 | 257 |
| 14.4 小结 | 262 |

第四部分：性能和扩展性

| | |
|----------------------------|------------|
| 第15章 性能调优 | 264 |
| 15.1 转换性能：找到最弱连接 | 264 |
| 15.1.1 通过简化找到性能瓶颈 | 265 |
| 15.1.2 通过度量值找到性能瓶颈 | 266 |
| 15.1.3 复制数据行 | 267 |
| 15.2 提高转换性能 | 269 |
| 15.2.1 提高读文本文件的性能 | 269 |
| 15.2.2 写文本文件时使用延迟转换 | 271 |
| 15.2.3 提高数据库性能 | 272 |
| 15.2.4 数据排序 | 275 |
| 15.2.5 减少CPU消耗 | 276 |
| 15.3 提高作业性能 | 280 |
| 15.3.1 作业里的循环 | 280 |
| 15.3.2 数据库连接池 | 281 |
| 15.4 小结 | 281 |
| 第16章 并行、集群和分区 | 283 |
| 16.1 多线程 | 283 |
| 16.1.1 数据行分发 | 284 |
| 16.1.2 记录行合并 | 285 |
| 16.1.3 记录行再分发 | 285 |
| 16.1.4 数据流水线 | 286 |
| 16.1.5 多线程的问题 | 287 |

| | |
|------------------------|-----|
| 16.1.6 作业中的并行执行 | 289 |
| 16.2 使用Carte子服务器 | 289 |
| 16.2.1 配置文件 | 289 |
| 16.2.2 定义子服务器 | 290 |
| 16.2.3 远程执行 | 291 |
| 16.2.4 监视子服务器 | 291 |
| 16.2.5 Carte安全 | 291 |
| 16.2.6 服务 | 292 |
| 16.3 集群转换 | 293 |
| 16.3.1 定义一个集群模式 | 293 |
| 16.3.2 设计集群转换 | 294 |
| 16.3.3 执行和监控 | 295 |
| 16.3.4 元数据转换 | 296 |
| 16.4 分区 | 298 |
| 16.4.1 定义分区模式 | 299 |
| 16.4.2 分区的目标 | 300 |
| 16.4.3 实现分区 | 300 |
| 16.4.4 内部变量 | 301 |
| 16.4.5 数据库分区 | 301 |
| 16.4.6 集群转换中的分区 | 302 |
| 16.5 小结 | 302 |

| | |
|-----------------------------|------------|
| 第17章 云计算中的动态集群 | 303 |
| 17.1 动态集群 | 303 |
| 17.1.1 建立动态集群 | 304 |
| 17.1.2 使用动态集群 | 306 |
| 17.2 云计算 | 306 |
| 17.3 EC2 | 307 |
| 17.3.1 如何使用EC2 | 307 |
| 17.3.2 成本 | 307 |
| 17.3.3 自定义AMI | 307 |
| 17.3.4 打包新AMI | 310 |
| 17.3.5 中止AMI | 310 |
| 17.3.6 运行主节点 | 310 |
| 17.3.7 运行子节点 | 311 |
| 17.3.8 使用EC2 集群 | 312 |
| 17.3.9 监控 | 313 |
| 17.3.10 轻量原则和持久性 | 314 |
| 17.4 小结 | 314 |

| | |
|--------------------------|------------|
| 第18章 实时数据整合 | 315 |
| 18.1 实时ETL介绍 | 315 |
| 18.1.1 实时处理面临的挑战 | 316 |
| 18.1.2 需求 | 316 |
| 18.2 基于流的转换 | 317 |
| 18.2.1 一个基于流的转换实例 | 318 |
| 18.2.2 调试 | 321 |
| 18.2.3 第三方软件和实时整合 | 321 |
| 18.2.4 Java 消息服务 | 322 |
| 18.3 小结 | 324 |

第五部分：高级主题

| | |
|--------------------------------|------------|
| 第19章 Data Vault管理 | 326 |
| 19.1 Data Vault 模型介绍 | 327 |
| 19.2 你是否需要Data Vault | 327 |

| | | | |
|--|------------|--------------------------------------|------------|
| 19.3 Data Vault的组成部分 | 328 | 21.6.1 RSS结构 | 396 |
| 19.3.1 中心表 | 328 | 21.6.2 Kettle对RSS的支持 | 398 |
| 19.3.2 链接表 | 329 | 21.7 小结 | 403 |
| 19.3.3 附属表 | 329 | 第22章 Kettle集成 | 404 |
| 19.3.4 Data Vault 特点 | 331 | 22.1 Kettle API | 404 |
| 19.3.5 构建 Data Vault 模型 | 331 | 22.1.1 LGPL协议 | 404 |
| 19.4 将Sakila的例子转换成Data Vault 模型 | 331 | 22.1.2 Kettle Java API | 405 |
| 19.4.1 Sakila中心表 | 331 | 22.2 执行存在的转换和作业 | 406 |
| 19.4.2 Sakila 链接表 | 332 | 22.2.1 执行一个转换 | 406 |
| 19.4.3 Sakila 附属表 | 333 | 22.2.2 执行一个作业 | 407 |
| 19.5 加载Data Vault 模型: 简单的ETL解决 方案 | 334 | 22.3 应用程序中嵌入Kettle | 408 |
| 19.5.1 安装Sakila Data Vault | 335 | 22.3.1 Pentaho 报表 | 408 |
| 19.5.2 安装ETL方案 | 335 | 22.3.2 把数据放到转换里 | 410 |
| 19.5.3 创建一个数据库账户 | 335 | 22.3.3 动态转换 | 413 |
| 19.5.4 ETL解决方案的例子 | 335 | 22.3.4 动态模板 | 416 |
| 19.5.5 加载 Data Vault 表 | 341 | 22.3.5 动态作业 | 416 |
| 19.6 从Data Vault 模型更新数据集市 | 341 | 22.3.6 在Kettle里执行动态ETL | 419 |
| 19.6.1 ETL解决方案例子 | 342 | 22.3.7 Result | 419 |
| 19.6.2 dim_actor 转换 | 342 | 22.3.8 替换元数据 | 420 |
| 19.6.3 dim_customer 转换 | 343 | 22.4 OEM版本和二次发布版本 | 421 |
| 19.6.4 dim_film 转换 | 346 | 22.4.1 创建PDI的OEM版本 | 421 |
| 19.6.5 dim_film_actor_bridge 转换 | 347 | 22.4.2 Kettle的二次发布 (Forking) | 422 |
| 19.6.6 fact_rental 转换 | 347 | 22.5 小结 | 423 |
| 19.6.7 加载星型模型里的所有表 | 349 | 第23章 扩展Kettle | 424 |
| 19.7 小结 | 349 | 23.1 插件架构 | 424 |
| 第20章 处理复杂数据格式 | 350 | 23.1.1 插件类型 | 425 |
| 20.1 非关系型和非表格式的数据格式 | 350 | 23.1.2 架构 | 425 |
| 20.2 非结构化的表格型数据 | 351 | 23.1.3 前提 | 425 |
| 20.2.1 处理多值字段 | 351 | 23.2 转换步骤插件 | 428 |
| 20.2.2 处理重复的字段组 | 352 | 23.2.1 StepMetaInterface | 428 |
| 20.3 半结构化和非结构化数据 | 353 | 23.2.2 StepDataInterface | 434 |
| 20.4 键/值对 | 358 | 23.2.3 StepDialogInterface | 434 |
| 20.5 小结 | 362 | 23.2.4 StepInterface | 440 |
| 第21章 Web Services | 363 | 23.3 用户自定义 Java 类步骤 | 444 |
| 21.1 Web 页面和Web Services | 363 | 23.3.1 传递元数据 | 444 |
| 21.2 数据格式 | 365 | 23.3.2 访问输入和字段 | 445 |
| 21.2.1 XML | 365 | 23.3.3 代码片段 | 445 |
| 21.2.2 HTML | 366 | 23.3.4 例子 | 445 |
| 21.2.3 JavaScript Object Notation | 367 | 23.4 作业项插件 | 446 |
| 21.3 XML例子 | 369 | 23.4.1 JobEntryInterface | 446 |
| 21.3.1 XML例子文件 | 369 | 23.4.2 JobEntryDialogInterface | 448 |
| 21.3.2 从XML中抽取数据 | 371 | 23.5 分区插件 | 448 |
| 21.3.3 生成XML文档 | 378 | 23.6 资源库插件 | 450 |
| 21.4 SOAP例子 | 384 | 23.7 数据库类型插件 | 450 |
| 21.4.1 使用“Web服务查询”步骤 | 385 | 23.8 小结 | 451 |
| 21.4.2 直接访问 SOAP服务 | 386 | 附录A Kettle生态群 | 452 |
| 21.5 JSON例子 | 389 | 附录B Kettle 企业版特性 | 456 |
| 21.5.1 Freebase项目 | 389 | 附录C 内置的变量和属性参考 | 457 |
| 21.5.2 使用Kettle 抽取Freebase数据 | 392 | | |
| 21.6 RSS | 396 | | |

第一部分：开始

本部分包括

第1章 ETL入门

第2章 Kettle基本概念

第3章 安装和配置

第4章 ETL示例解决方案——Sakila

CHAPTER

1

第1章 ETL入门

本章的内容是数据整合工作的起点，本章将详细解释3种主要的数据整合方式的不同点和相似点。这3种数据整合方式分别是ETL、ELT和EII。为了能够全面理解数据仓库和数据整合，我们先来看看事务型数据库系统和分析型数据库系统的不同之处。

1.1 OLTP和数据仓库对比

人们通常问的第一个问题是事务系统和商业智能（Business Intelligence，简称BI）系统有什么区别（商业智能系统通常也被称为决策支持系统 Decision Support System，简称DSS）。一个独立的事务系统通常也被称为在线事务处理系统（OnLine Transaction Processing，简称OLTP），事务系统需要能够快速地定位到一条记录。当一次需要获取多条记录时，多条记录通常使用唯一的键值加以识别。例如订单系统中的一个订单信息，人力资源系统中的一个人信息。更重要的是这些信息需要经常被更新，通常一次只更新一条记录。

OLTP和BI数据库（Data Warehouse，简称DWH）的最大区别是在一个单一事务里要分析的数据的数量。OLTP系统中，很多并发的用户请求通常只处理一条数据或有限的一组数据，而数据仓库系统必须有处理几百万条数据的能力，来响应用户的一个简单请求。表1-1显示了OLTP系统和数据仓库系统的主要区别。

表1-1 OLTP和数据仓库对比

| 指标 | OLTP | 数据仓库 |
|--------|----------|--------|
| 系统覆盖范围 | 单一业务处理系统 | 多个业务主题 |
| 数据源 | 一个 | 多个 |

续表

| 指标 | OLTP | 数据仓库 |
|------------|---------|-----------|
| 数据模型 | 静态 | 动态 |
| 主要查询类型 | 插入/更新 | 只读 |
| 单事物数据量 | 小 | 大 |
| 数据量 | 小/中 | 大 |
| 数据时间精确度 | 当前时间戳 | 从秒到天不等 |
| 批量加载/插入/更新 | 否 | 是 |
| 全部历史数据访问性 | 否 | 是 |
| 响应时间 | <1秒 | <10秒 |
| 系统可达性 | 7×24 小时 | 5×8 小时 |
| 典型用户 | 前端业务用户 | 分析人员、决策人员 |
| 用户数量 | 大 | 小/中 |

当然表中所列出的指标并不是绝对的，而是区分两类系统的传统经典的方式。在实际中，BI系统也越来越多地被作为业务系统的一部分。例如在呼叫中心系统中，客服人员前面的屏幕上不但能显示出客户的姓名、地址等基本属性，也能显示出客户的历史交易记录，这些交易记录一般是从业务数据存储系统（Operational Data Store，简称ODS）或数据仓库系统里获得的。很多客户关系管理（简称 CRM）系统也能即时地获得客户信用或等级，这些信用或等级数据也是从数据仓库中经过预处理获得的，提供给前端的业务操作人员。这说明数据仓库渗透到业务系统的程度越高，对数据仓库的要求也越接近于对业务系统的要求，尤其是系统可达性和数据时间精确度方面的要求。

对数据仓库最多的讨论可能就是响应时间，在十年前花一两分钟的时间抽取并展现数据可能不算大问题。今天用户希望的响应时间是他们已经习惯了的搜索引擎的响应时间。超过10秒钟用户就没有了耐心，开始单击刷新按钮（这种操作往往会加重系统负担，使情况变得更糟），最终因为响应时间太长了，而不再使用数据仓库。而另一方面，如果数据仓库用于数据挖掘目的，只要最终的挖掘结果是有价值的，即使响应时间有几个小时也是可接受的。

1.2 ETL是什么

你肯定知道ETL是抽取、转换、加载的缩写。但ETL的确切含义是什么？这里有一个ETL的简单定义：“将数据从OLTP系统中转移到数据仓库中的一系列操作的集合”。从ETL的根源来看，可以接受这个定义，但从ETL实践角度来看，这个定义过于简单。在实际情况中，数据不仅仅来源于OLTP系统，还来源于网站、平面文件、电子邮件系统、电子表单、还有像Access这样的个人数据库。而且ETL不仅仅用来将数据加载到数据仓库，还可以有其他用途，如加载数据集市、生成电子表格、使用数据挖掘模型为客户分级，甚至将预测数据回写到OLTP系统。尽管ETL应用范围广泛，但一般来说还是可以分为以下3个部分。

1. 抽取：一般抽取过程需要连接到不同的数据源，以便为随后的步骤提供数据。这一部分看上去简单而琐碎，实际上它是ETL解决方案成功实施的一个主要障碍。
2. 转换：在抽取和加载之间的，任何对数据的处理过程都是转换。这些处理过程通常包

括（但不限于）下面一些操作。

- 移动数据
 - 根据规则验证数据
 - 数据内容和数据结构的修改
 - 集成多个数据源的数据
 - 根据处理后的数据计算派生值和聚集值
3. 加载：将数据加载到目标系统的所有操作。在第5章会说明加载并不仅仅是将数据批量装载到目标表。加载过程还包括对代理键的管理和对维度表的管理等。

本章的剩余部分说明了ETL解决方案的演化过程以及主要的ETL构建模块。

1.2.1 ETL解决方案的演化过程

自从数据以电子化形式保存以来，就有了数据整合的需求。在数据整合早期，ETL工具出现之前，人们使用手工编写程序的方式来完成不同数据源的数据整合工作，如早期的COBOL、RPG和后来的Perl或PL/SQL。尽管这只是第一代的数据整合方案，但直至今天仍有45%的ETL工作使用这种手工编程/脚本的方式来完成。相对价格昂贵的ETL工具（通常价格是六位数）而言，手工编程还有一定意义，但是现在已经有了越来越多的开源或低价的ETL工具，再使用手工编程方式完成ETL工作就已经没有太多的意义了。手工编程的主要缺点在于：

- 容易出错
- 开发周期长
- 不易于维护
- 缺少元数据
- 缺乏一致性的日志和错误处理

第二代的ETL工具（这里我们还是称其为“工具”，而不是广义上的“解决方案”）试着克服这一问题，方法是依据设计好的ETL workflow来自动生成所需代码。在20世纪90年代初，出现了Prism、Carlton，以及ETI这些产品，但是大多数产品后来都被其他ETL供应商收购了。ETI大概是最后剩下的唯一一个提供代码生成的解决方案独立供应商。在目前的第二代的ETL解决方案中仍然具有代码生成技术，这丝毫不意味代码生成功能已经过时了。恰恰相反，代码生成技术正生机勃勃。例如Oracle的数据仓库产品毫无疑问是这个领域最负盛名的产品，同样的，流行的开源工具Talend也是代码生成解决方案的另一个例子。

代码生成也有利有弊，最大的弊端是大多数代码生成仅能用于有限的特定数据库。不久之后，就在代码生成技术广泛应用之时，第三代ETL工具出现了。第三代ETL工具采用基于引擎的架构，可以执行所有的数据处理进程，还可以将所有的数据库连接和转换规则作为元数据存储起来。因为引擎有标准的工作方式，所有的转换在逻辑上是独立的，无论是相对于数据源还是数据目标。基于引擎的ETL工具通常比代码生成的方式更具通用性。Kettle就是一个基于引擎ETL工具的典型例子。在这个领域，还有一些其他熟悉的名字，比如Informatica Powercenter以及SQL Server Information Services。

无论是代码生成器还是基于引擎的工具，都能帮助我们发现数据源的底层架构，以及这些架构之间的关系，当然在这些工具里有一些工具比另一些工具在这方面做得更好。但它们都需

要开发目标数据模型，或者先行开发，或者在设计数据转换步骤时开发。设计阶段过后，还必须进行目标数据模型与源数据模型的映射。而整个过程是相当耗时的。所以到了最后，新一代的模型驱动的数据仓库工具随之出现了。MDA（即模型驱动的架构）工具试图自动化实现数据仓库和数据集市的设计过程，读取源数据模型，生成目标数据模型与需求数据之间的映射，以便向目标表填充数据。目前市面上只有其中一小部分的工具，这其中Kalido和BIReady最广为人知。然而，世上没有万能的解决方案，MDA工具仍然需要具备一定的技能的数据仓库开发人员才能发挥作用。虽然它们不能解决所有的数据集成问题（尤其有挑战性的问题），但还是可以节省大量的时间（甚至金钱）。

数据仓库与数据集市

在本书中，数据仓库和数据集市似乎是两个通用的词。然而事实并非如此，这两个词在使用范围、模型和适用性方面有着巨大的差异。数据仓库是单一的，大量（历史性）数据的存储仓库，可用来支持企业决策。因此，它所涉及的数据涵盖了各种主题和各种业务领域，例如金融、物流、市场营销和客户支持。通常，一个数据仓库不能被终端用户工具直接访问。相反，一个数据集市可以由终端用户直接访问，并且是以特定的数据分析为目的的，例如零售或客户来电。

1.2.2 ETL基本构成

ETL解决方案就像一个业务流程。业务流程具有输入、输出，以及一个或多个工作环节，处理步骤。同样的，这些步骤也具有输入和输出，并可以执行将一个输入转化为输出的操作。想一想，例如，在一家保险公司理赔部，门上有一个大牌子，上面写着理赔部，这就意味着它描述了部门的主要职责和业务：处理理赔。而在部门里，你会发现每张办公桌上或分部门可能有其自身的特点：健康保险理赔、汽车保险理赔、旅游保险理赔，等等。当接纳一个理赔案件时，首先确定这个理赔将被哪个部门处理。然后部门办公人员根据是否有提供理赔必需的信息来决定是否处理它，如果不符合的话，退回给提交者，并且给予说明。理赔处理的工作时间是每天早上9点到下午5点。

这个例子和ETL处理过程非常相似：首先一个验证步骤去确定到达的或者被抽取的数据是哪一种类型，然后数据被送到一个特定转换去处理。当转换执行完后，数据将被传递到下一个转换或者一个目标表，并在发生错误的情况下，被转移到一个错误处理流程进行处理。每个晚上凌晨3点，一个调度程序开始此项任务并且直到所有数据被处理才结束。

你现在可能对设计ETL处理流程有一个整体的认识了。从前面的例子可以推断出，必须有某种机制来控制整个处理流程，以及实际转换的细节工作。用Kettle的术语阐述的话，前面部分称为作业（job），后面部分称为转换（transformation）。作业是ETL解决方案的代理，而转换是基础的构建部分。独立的转换能够被链接在一起形成一个具有逻辑顺序的队列，形成一个能被调度和执行的作业，就像一个业务流程。同样的，转换也是由几个步骤组成的。步骤是Kettle解决方案的第三种基本构成块，而步骤之间的连接关系由跳（hop）来决定。在本书的其他部分，你将会更多地了解作业、转换、步骤以及跳的概念。这4个基础组成部分使你能够开发任何的ETL解决方案。第2章对这4个概念提供了更详细的介绍。

1.3 ETL、ELT和EII

数据整合是一个比ETL更加广泛的概念，ETL是指从一个或多个数据源抽取数据，经过一个或多个转换步骤后，物理地存储到目标环境中，目标环境通常是数据仓库。为了能把ETL和其他数据整合方式区分开，我们需要一种分类和描述方法。

图1-1是一个数据仓库架构的典型例子。在图中有多个业务源系统，一个数据中转区，一个保存了所有历史数据的数据仓库和多个可以由终端用户访问的数据集市。这些组成部分之间都是由数据整合过程来完成的，就是图中显示的ETL部分。

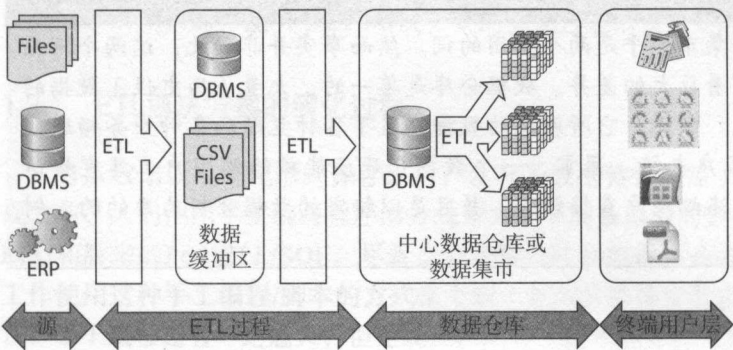


图1-1 典型的数据仓库架构

这个架构在过去20年里一直使用得很好，现在很多数据仓库项目也都在使用类似的架构。图1-1中显示出ETL工具不但用于抽取数据并加载到数据仓库里，而且用于加载数据集市或者其他业务系统的数据库（图中未显示）。

在图1-1中还可以看到在源系统和数据仓库之间还有一个数据中转区或数据缓冲区（staging area），数据中转区仅用来快速地从源数据系统中获取数据，并暂时保留这些数据。数据中转区不一定是一个数据库系统，在很多情况下，将数据保存在ASCII文件中比插入数据库表中更快。

ETL工具的其他用途

ETL工具并不仅仅在数据仓库中使用，由于这些工具都提供了各种类型的连接和转换选项，因此另一个经常使用到的应用为数据迁移，在类似于Kettle的ETL工具协助下，可以非常轻松地连接A数据库并将其数据迁移至B数据库。事实上，Kettle拥有两种向导（复制表和复制多表），可以根据目标数据库的SQL语法生成新的目标表，使得你的数据迁移变得自动化。

第三个较为复杂的应用为数据同步，即利用ETL工具保持两个（或两个以上）数据库之间的同步。尽管这样做可以达到一定的效果，但这并不是ETL最常用的使用方式。通常情况下同步有时间限制（当A数据库发生变化时B数据库需要在最短的时间内响应），而大多数ETL工具面向批量数据处理，因此并不完全适用于同步的场景。

1.3.1 ELT

ELT（抽取、加载和转换的简称）同ETL在数据整合的方法上略微不同。在ELT的情况下，数据首先从源数据（可能是多个）进行抽取、加载到目标数据库中，再转换为所需要的格式。所有大数据量处理全部放在目标数据库中进行。这种做法的好处在于，一般情况下，数据库系

统更适合处理负荷在百万级以上的数据集成。数据库系统也通常会对I/O（吞吐量）进行优化，用来提高数据处理速度。

有一个与ETL比较大的区别在于：ELT工具需要知道如何使用目标数据库平台和相应的SQL方言。这就是在市面上ELT解决方案较少的原因，类似Kettle这样通用的ETL工具也同样缺少这些功能。然而，大部分闭源ETL厂商都声称他们的产品支持本地SQL（pushdown SQL）的功能，也就是支持ETLT（抽取、转换、加载、转换）的场景，在ETLT场景下，既可以由转换引擎执行转换（特别是针对目标数据库不支持的操作），也可以由数据库执行转换。主流数据库厂商如微软（SQL Server Integration Services）和甲骨文（Oracle Warehouse Builder），由于他们的ETL工具已经同本身的数据库产品紧密地捆绑在一起，因此在ETLT的设计上处于领先地位。甲骨文不久前收购了一家叫作Sunopsis的公司，这家公司有市场上为数不多的ELT的解决方案（就是现在的Oracle Data Integrator，简称ODI），另外，类似于Informatica和Business Objects也在其最新的产品中增加了pushdown SQL的功能。Dan Linstedt的博客详细描述了ETL和ELT的优缺点，该作者是Data Vault data warehouse 建模技术的发明者，可参考http://www.b-eye-network.com/blogs/linstedt/archives/2006/12/etl_elt_-_chall.php。

这里要说一个特殊的数据库产品LucidDB，这个面向列存储的开源BI数据库将ETL和ELT概念更往前进了一步。它通过使用标准ANSI SQL的扩展，可以在数据库内部完成所有的ETL功能。LucidDB可以对不同数据源（数据库、文本文件、WebService）进行封装，对数据源封装后，就可以使用标准的SQL来访问数据源。这种架构可以使LucidDB作为轻量级的EII（Enterprise Information Integration）解决方案，我们将在接下来的部分讨论EII。

1.3.2 EII：虚拟数据整合

ETL和ELT都是以物理方式将数据从OLTP移动或复制到数据仓库。在前面的“OLTP和数据仓库对比”部分中已经说明了为什么要使用独立的数据仓库。但是在越来越多的情况下，没有必要移动或复制数据。实际上，大多数用户并不关心ETL过程和数据仓库：他们只是想获得他们想要的数！某种意义上，图1-1显示的数据仓库架构就像饭店的厨房。作为一个顾客我并不知道饭菜是如何做出的，我只是希望能准时并且味道不错。厨房里发生的事情则和我无关。这个道理也同样适用于数据仓库：用户并不关心数据是如何处理的；他们只是想快速而容易地访问到数据。

除了物理数据集成方式，还有虚拟数据集成方式也可以满足用户访问数据的要求。这种虚拟数据集成方式就是企业信息集成，也就是EII。还有其他一些名称，如数据联邦和数据虚拟化，也都是描述同样的事情。这种方法的主要优点就是数据永远都是最新的。另一个优点就是不需要额外的存储层，没有冗余数据。在数据仓库环境下，同一数据可能要保存三到四份，一份在数据中转区，一份在业务系统，还有一份在数据仓库或数据集中。通过使用虚拟数据集成技术，业务系统的数据可以像数据集市一样由用户访问。EII工具在后台完成了所有的翻译和转换工作。

尽管EII听上去不错，但它也有一些缺点。表1-2说明了物理和虚拟这两种集成方式的区别。从表1-2可以得出一些结论，结论之一就是使用虚拟方法来管理大量的、清洗后的、时效性强的数据是一个非常具有挑战性的工作。另一个结论就是ETL工具只属于物理集成的范畴，在第22章我们将看到Pentaho报表在生成即席报表时可以将Kettle转换作为一个数据源。这种方法实际上就

是将虚拟方案的优点和ETL工具的丰富功能结合起来。

表1-2 虚拟和物理数据集成比较

| 特征 | 物理 | 虚拟 |
|-----------|----|----|
| 数据时效性 | ○ | ● |
| 查询性能/延迟 | ● | ⊙ |
| 访问频率 | ● | ⊙ |
| 数据源的多样性 | ⊙ | ● |
| 数据类型的多样性 | ⊙ | ● |
| 非关系型数据源 | ○ | ● |
| 转换和清洗 | ● | ○ |
| 性能可预测性 | ● | ⊙ |
| 同一数据的多重接口 | ○ | ● |
| 大批量查询 | ● | ○ |
| 历史数据/聚合 | ● | ○ |

表中图示说明：○弱，⊙一般，●强

Mark Madsen, Third Nature, Inc., 2009. 保留所有版权，允许下使用。

1.4 数据整合面临的挑战

数据整合往往面临着一系列挑战。这些挑战可能带有政治的、组织性的、功能性的或者技术性的特征。

首先，往往也是最重要的，你需要明确解决业务问题和发布企业信息都需要哪些数据。如果BI项目不去解决具体的业务需求，项目可能得不到必要的资助。技术上的障碍也是一种挑战，但是在大多数情况下都是可以解决的；组织和协调方面的障碍则更难解决。我们虽然不会在本书中深入探讨这些障碍，但要提醒大家意识到存在的这些障碍。

注意：更多内容可以参考 Ralph Kimball的*Data Warehouse Lifecycle Toolkit*（第2版）的第3章 addresses gathering business requirements中的内容。

要启动一个BI项目需要一个可行的项目计划和实际的业务场景，但只有上述两项并不能够保证项目成功实施。要使项目成功，一套成熟的方法必不可少，当然一个富有经验的团队也更有助于项目的成功。多年来IT项目都是按照瀑布模型来运行的，也就是按照初始需求阶段、设计阶段、开发阶段、测试阶段进行，然后部署到生产环境中。对于BI项目来说，ETL是一个很重要的部分，但很难做得完美。下面部分讲述的敏捷方法可以更好地完成BI项目里的各个环节。

另外，你还面临着ETL设计的挑战，如何定义作业和转换，这不是纯技术层面的工作，而是功能层面的工作。使用ETL解决一个实际问题可能有很多方法，但无论使用哪种方法，在概念上应该是一致的。例如，若开发团队决定使用文件作为数据中转区，那么就要坚持使用文件，不要混合使用文件和数据库作为数据中转区，除非绝对的万不得已。

在解决了组织性的、项目方面和设计方面的挑战后，遇到的第一个技术问题就是从哪里可以找到需要的数据，以何种格式获得数据，数据都包括了哪些内容。即使知道了数据在哪里，或

知道了数据的格式，但如何连接到并获取这些数据也可能是一个问题。很多数据可能位于企业信息系统的主节点上，或者其他历史遗留的UNIX系统里。

大数据量也是一个挑战，在大多数场景下，不可能每次ETL过程都抽取全部数据。所以你能跟踪到源数据系统内发生的变化，以便以增量的方式抽取到增加、删除、修改的数据。在一些场景下增量抽取问题并不能完美解决，可能会采用暴力比较的方法来比较源系统和数据仓库里的数据，以获得增量数据。

根据数据集成的方式的不同，可能还会有其他一些挑战。想象一下有三个不同的业务系统，每个业务系统里都有一部分客户数据，要将这三个系统里的用户数据合并加载到数据仓库里，如何解决数据的不完整、不一致，或者丢失的数据？

1.4.1 方法论：敏捷BI

任何项目面临的第一个问题就是要找到一个好的开发和交付实施的方法，也包括相关文档的交付。不只对于ETL，对于任何软件开发都是如此。在过去几年里，已经有一些好的方法应用于项目管理和软件开发。或许你还记得结构分析和设计等一些方法论，这些方法论是在上世纪70年代由Ed Yourdon和Tom DeMarco等人发展起来的。这些方法都有一个所谓的瀑布模型，瀑布模型强调了在分析或者设计阶段的一个步骤完成以后，才能开始下一个步骤。关于瀑布模型相关方法的更多信息，可以参考：http://en.wikipedia.org/wiki/Structured_Analysis。

在上世纪80年代和90年代，软件开发者们发现这些结构化的瀑布模型方法并不能适用于所有场景，尤其在需求经常变化的情况下。为了解决这种需求经常变化的问题，出现了各种不同的“敏捷”开发方法。在这些方法中，也许Scrum（译者注：Scrum是1986年由Hirotaka Takeuchi和Ikujiro Nonaka提出的软件开发方法，Scrum原意为橄榄球比赛中的争球，这里将橄榄球比赛中球传来传去的过程类比软件开发中在多个阶段反复迭代的过程，用来强调速度和灵活性）方法是最著名的一个。敏捷开发方法到底有什么不同？为了澄清一些观点，敏捷方法的创立者和支持者签署了敏捷宣言（译者注：敏捷宣言是2001年2月13日由17位软件专家在美国犹他州通过两天的讨论通过的），敏捷宣言中定义了敏捷方法的4条价值观：

- 个体与交互胜过过程和工具
- 可用的软件胜过完备的文档
- 客户协作胜过合同谈判
- 响应变化胜过遵循计划

敏捷宣言（全文见 <http://agilemanifesto.org>）同时还定义了12条指导原则。简短来说，这12条原则的主要含义是：

- 更早更快交付可用软件。
- 欢迎需求的变更。
- 业务人员和IT人员紧密地在一起工作。
- 依赖于自驱动的开发者 and 自组织的团队。
- 要经常面对面地讨论问题和过程。
- 简捷，即尽最大可能减少不必要的工作。

注意: 访问[http://en.wikipedia.org/wiki/Scrum_\(development\)](http://en.wikipedia.org/wiki/Scrum_(development))可以查询到更多和敏捷开发以及Scrum方法相关的内容。

那么敏捷开发方法和商业智能,特别是ETL有什么关系?

对于Pentaho和Kettle而言,这种关系是全方位的! Pentaho 本身就使用了敏捷开发方法(尤其是Scrum方法),以增量迭代的方式来开发和发布BI平台及组件的新版本。开发中的不同阶段是以Sprint和里程碑来度量的,这一点可以从下载的软件版本中看出,可以从CI(持续集成)资源库中下载软件版本。2009年, Pentaho 公司决定把开发Pentaho BI套件过程中使用的敏捷开发经验提升为敏捷BI方法。这样做不仅可以给BI 开发者提供一个有效的开发方法,而且可以改进Pentaho BI 组件,使之可以用于或支持敏捷的工作方式。Kettle是BI 套件里第一个发生变化的组件,这也是我们要在本书中介绍敏捷概念的原因。Kettle的敏捷能力本书随后会有更多讲解,在这里我们先看看Kettle 提供的基本敏捷能力。

安装完Kettle 后,只要用上几分钟的时间就可以完成从创建数据源、创建转换、抽取数据,到加载数据到目的表的过程。因为Kettle是基于引擎的解决方案,它会自动为你处理很多事情,这样可以加快数据处理过程。Kettle包括了大量的组件和转换步骤(第一眼看上去可能令人眼花缭乱)。这些内置的组件可以将开发工作量最小化,也将解决方案交付的速度最大化。在数据处理流程中,前面步骤如果改变了数据字段或数据类型,这种变化会自动传递到随后的步骤,并自动生成修改目标表的Alter语句脚本。这种可视化的转换模型可以将你设计的转换直接展示给用户并以一种交互的方式进行数据处理。因此开发人员和业务人员能使用同一个工具并紧密地在一起工作。这样如果开发过程中设计的转换偏离了用户的期望,也可以被快速地更正过来。

因为Kettle是与 Pentaho 提出的敏捷BI方法紧密结合的,所以我们有必要从这个链接<http://wiki.pentaho.com/display/AGILEBI/Welcome+to+Agile+Business+Intelligence>中详细了解敏捷BI的方法论,以及Kettle如何支持敏捷BI。

1.4.2 ETL设计

即使是(或者说特别是)采用了敏捷开发方法,你也要用多种不同方式来设计一个ETL解决方案。在很多方面,ETL解决方案更像一个 workflow 或业务流程。在开始构造ETL过程之前,应该在一个比较高的层次上使用流程图工具来绘制一个流程图。大多数用户都熟悉流程图,可以通过注释使流程设计得更具有可读性。在较细节的层次上,还要定义好转换的哪一部分可以重用。例如,在数据仓库项目里创建日期维度往往是一次性的工作,所以没有必要花大量时间在一个单独的项目里设计一个非常灵活的与数据库独立的可复用的日期维度转换。但是如果你是一个顾问或团队,经常要为不同的公司服务,面临着不同数据库环境,那么就要创建一个灵活的、与数据库独立的、通用日期维度生成器,作为你的常备工具。

从上述讨论可以看出最重要的问题是,当开始构造数据转换时要考虑到转换是否要在其他方案里重用? 如果重用,你就要花更多时间使转换更有通用性,例如在转换里增加额外的参数以设置不同的时间、数据库类型、连接参数等这些会发生变化的值。

1.4.3 获取数据

正如前面讨论的那样,如何从源系统中获取数据是ETL项目开始后面临的第一个问题。不要

仅仅把获取数据问题作为一个技术问题，在很多情况下，由于企业内部的政策或制度问题使得我们不能获取数据。一些 ERP 系统软件的供应商也在尽力使自己系统内的数据不被其他系统访问，或者根本禁止其他系统的直接访问。例如广泛使用的 SAP/R3 系统，在软件许可（License）中有一个特别的条款，该条款禁止除 SAP 提供的方法以外的任何方法直接连接底层数据库。大多数金融机构在主节点上运行的核心系统，也不会让你直接连接到底层数据库系统，所以你就会依赖于 FTP、Web Services 这样的提供数据的方式。其实不能直接访问数据源，也不一定是个缺点，SAP 系统里包括了超过 70 000 张表，想从这里找到一个你需要的表，也是一件棘手的事情。在这种情况下，就需要一些工具可以将 ERP 系统里的元数据转换成数据的业务视图。Kettle 包括了一个标准输入步骤，可以从 SAP/R3 中获取数据，还包括了一个标准输入步骤可以从 Salesforce.com 中获取数据，Salesforce.com 可能是现在最流行的在线 CRM 系统了。对于其他标准 ERP 和 CRM 系统来说，如 SugarCRM、OpenERP、ADempiere 或 Peoplesoft，你可能需要第三方的解决方案，或者自己开发一个输入步骤插件。在不能从数据源直接获取数据的情况下，最好要求数据源以一种可读编码格式，如 ASCII 或 UniCode（一些旧的系统还在使用 EBCDIC 编码），来提供数据。关于访问历史遗留的 Cobol 系统和其他一些特定格式文件的相关问题，可以参考下面的链接：<http://jymengant.iframe.com/jymengant/jurassicfaq.html>。

电子表格问题

在数据获取方面还有一个很大的问题就是数据的格式。在这个领域，最恶名昭著的恐怕就是 Excel 了。最好的建议就是不要接受以 Excel 格式提供的数据，除非你确认 Excel 文件是系统自动生成的而且生成 Excel 的机器和运行转换的机器属于同一个用户。当使用了不同的国际化的设置，Excel 的数据格式问题更是层出不穷，导致日期或数值类型数据的格式在不同会话中不一致。结果你精心设计的转换可能会运行失败，或者产生了不正确的结果（这是两个类似的结果，但后者更可怕，因为难以追踪）。实际上，大多数的问题是用户引起的，不过用户会告诉你什么也没有动过，那么是谁动的呢？可能用户根本不知道自己动了什么。

失败处理

即使访问数据源本身不成问题而且你构建的作业流程看上去也很稳定，在实际转换任务开始前，你也要确保数据源是能访问到的。一个基本的设计原则就是即使数据源没有成功连接，你的 ETL 作业也应该是提示错误后退出，而不是异常退出。在这方面 Kettle 也提供了很多功能，你可以：

- 测试资源库是否可以连接。
- Ping 一台主机检查是否可连通。
- 以返回的行数为条件判断 SQL 语句运行成功或失败。
- 检查文件目录是否为空。
- 检查文件、数据库表、表的列是否存在。
- 文件和目录的比较。
- 设置 FTP 和 SSH 连接超时。
- 每一个作业项执行后，都有成功或失败的输出。

还有一个方法就是在作业和转换上增加错误处理。当加载数据仓库时，表之间经常存在着

一定的依赖性，例如所有维度表加载后才能加载事实表。当一个维度表加载失败时，全部的作业都要失败。一个好的设计可以在错误发生后，让你更正错误，重新启动作业后只执行失败的部分和没有执行的部分。

增量数据捕获CDC

ETL的第一步就是从不同的源系统中抽取数据，把数据传递到流程的下一个步骤。在数据仓库实践中，一个比较好的方法就是把从源系统中抽取出的数据放在缓冲的数据库表或文件中作为中间存储层（这个作为中间存储层的数据库表或文件就是数据缓冲区，Staging Area）。这样重新开始的ETL过程就不用再从源系统中获得数据，而可以直接从缓冲区的数据表获得。这看上去像是一个比较琐碎的问题，但在数据仓库初始化阶段，除了数据量问题和低速的网络连接问题之外，就要注意这个问题了。

在数据最初始的加载完成后，你肯定不想再重复一次数据完全加载的过程。你已经有了一个差不多的数据集，只需要再刷新一下，以获得最新数据。你感兴趣的只是自上次抽取后又有那些数据发生了变化，所以想要知道哪些数据被增加、修改，甚至删除。辨别出哪些数据发生变化，并抽取那些自上次抽取后发生变化的数据的过程就被称为增量数据捕获，也叫CDC。

在CDC 处理方式上有两种最基本的分类，侵入式（intrusive）和非侵入式（non-intrusive）。侵入式是指CDC 操作会对源系统有一定性能影响。可以这么说，不论以何种方式，只要对源系统执行了SQL 语句，就是侵入式的。不幸的是大多数的CDC方法都是侵入式的方法，只有一种方法不是侵入式。CDC 将在第6章中详细介绍。

1.4.4 数据质量

在设计数据转换流程时，要提前设想到你的数据会有质量问题，因此要考虑到如何处理这些问题。实际上数据质量问题应该在源系统中解决而不是在ETL过程解决。但是在开始数据仓库项目之前就修改数据质量问题，是一件耗时耗力的事情，大多数机构都承受不起。而在项目开发过程中，由于项目有时间的限制，即使在开发中发现了一些严重数据质量问题，也可能会延期处理这些问题，或根本不处理。因此，作为一个ETL开发人员，应该在作业描述里详细说明数据可能有哪些问题，应该如何解决。

有两类工具可以处理数据质量问题。第一类，使用数据分析工具，分析并调查数据的实际质量如何，并定义一个数据的基线版本。分析的目的有两个：一是可以将分析的结果反馈给数据的拥有者（希望是有对数据修改权限的业务经理），另一个是作为ETL作业中数据检验步骤的输入。第二种，使用数据质量工具，可以基于业务规则和质量规则长期地监控并修订数据。对Kettle来说，Kettle 里的数据检验步骤就是内置的数据质量工具。

数据分析

开始ETL项目的第一件事情就是分析源系统中的数据。通过分析可以从技术上和统计上知道数据有多少、数据是什么样子。最常见的分析就是列分析，表里的每一列都会有统计信息。根据列的数据类型的不同，可以获得下面一些分析结果：

- NULL或空值的个数。

- 不同的值的个数。
- 最小、最大、平均值（数值字段）。
- 最小、最大、平均长度（字符串字段）。
- 正则表达式（例如，电话号码字段####-####-####）。
- 数据的分布。

尽管大部分这些操作都可以使用Kettle 转换或SQL语句完成，但最好还是使用专门的工具来完成，如eobjects公司提供的数据清洗器。第6章将详细描述数据分析。

警告：数据分析也就只能做这些了，大多数的数据分析工具都不能完成逻辑的和/或跨系统的质量分析问题，如果想要分析这些问题，还需要先建立起全局的业务字典和元数据系统。这样的系统目前还非常少。

数据检验

数据分析通常用于报告和设置数据基线版本，而数据检验则是ETL工作的一部分。如下面的一个简单例子：技术上源系统的一些列可以包括NULL值，但是业务上需要这一列有值。数据分析可以发现有几条记录在这一列上包含了NULL 值。检验步骤就可以解决这样的问题：检验步骤可以为该列设置 NOT-NULL 检验，当列中包括了一个NULL 值，就可以触发某个动作，如忽略这条记录并把这条记录写到错误日志表里，或使用默认值替换这个NULL 值，或将记录标识为无效，或任何必要的其他动作。

1.5 ETL工具的功能

尽管本书是讲Kettle的，但我们也有必要了解一般ETL工具必备的特性和功能。这样可以判断出Kettle是否适用于你手边的工作。下面的每一部分都先描述了ETL工具的通用功能，接着再描述Kettle如何提供这些功能。

1.5.1 连接

任何ETL工具都应该有能力连接到类型广泛的数据源和数据格式。对于最常用的关系型数据库系统，还要提供本地的连接方式（如对于Oracle的OCI），ETL应该能提供下面最基本的功能：

- 连接到普通关系型数据库并获取数据，如常见的Oracle、MS SQL Server、IBM DB/2、Ingres、MySQL和PostgreSQL。
- 从有分隔符或固定格式的ASCII文件中获取数据。
- 从XML文件中获取数据。
- 从流行的办公软件中获取数据，如Access数据库和Excel电子表格。
- 使用FTP、SFTP、SSH方式获取数据（最好不用脚本）。

除了上述这些功能，还要能从Web Services或RSS中获取数据。如果还需要一些ERP系统里的数据，如Oracle E-Business Suite、SAP/R3、PeopleSoft或JD/Edwards，ETL工具也应该提供到这些系统的连接。

Kettle也提供Salesforce.com和SAP/R3的输入步骤,但不是套件内,需要额外安装。对于其他ERP和财务系统的数据抽取还需要其他解决方法。当然,最通用的方法就是要求这些系统导出文本格式的数据,将文本数据作为数据源。

1.5.2 平台独立

一个ETL工具应该能在任何平台上甚至是不同平台的组合上运行。一个32位的操作系统可能在开发的初始阶段运行很好,但是当数据量越来越大时,就需要一个更强大的操作系统。另一种情况,开发一般是在Windows或Mac机上进行的,而生产环境一般是Linux系统或集群,你的ETL解决方案应该可以无缝地在这些系统间切换。

1.5.3 数据规模

数据规模也是一个比较大的问题,你的解决方案应该能处理逐年增长的数据。一般ETL能通过下面3种方式处理大数据。

- **并发**: ETL过程能够同时处理多个数据流,以便利用现代多核的硬件架构。
- **分区**: ETL能够使用特定的分区模式,将数据分发到并发的数据流中。
- **集群**: ETL过程能够分配在多台机器上联合完成。

一些商业ETL工具按照CPU或服务器发放License,对于这样的产品,第三种方式会极大增加成本。

Kettle是基于Java的解决方案,可以运行在任何安装了Java虚拟机的计算机上。转换里的每个步骤都是以并发的方式来执行,并且可以执行多次,这样加快了处理速度。Kettle在运行转换时,根据用户的设置,可以将数据以不同的方式发送到多个数据流中(译者注:有两种基本发送方式,即分发和复制,分发类似于发扑克牌,以轮流的方式将每行数据只发给一个数据流,复制是将一行数据发给所有数据流)。为了更精确控制数据,Kettle还使用了分区模式,通过分区可以将同一特征的数据发送到同一个数据流。这里的分区只是概念上类似于数据库的分区,Kettle并没有针对数据库分区有什么功能。(ETL工具的这个功能也有些争议,一般认为数据库应该比ETL更适合来完成数据的分区。)

最有效的规模扩展的方式可能就是集群了,集群可以使Kettle将工作负载按需分配到很多机器上。本书的第4部分深入讲解了所有规模扩展的方式。更多规模扩展相关的内容可以参考Bayon Technologies的Nicholas Goodman写的白皮书,他同时也是本书的审阅者。参考http://www.bayontechnologies.com/bt/ourwork/pdi_scale_out_whitepaper.php。

1.5.4 设计灵活性

一个ETL工具应该留给开发人员足够的自由度来使用,而不能通过一种固定的方式来限制用户的创造力和设计的需求。ETL工具可以分为基于过程的和基于映射的。基于映射的工具只在源和目的数据之间提供了一组固定的步骤,严重限制了设计工作的自由度。基于映射的工具一般易于使用,可快速上手,但是对于更复杂的任务,基于过程的工具才是最好的选择。使用像Kettle这样基于过程的工具,根据实际的数据和业务需求,可以创建自定义的步骤和转换。

1.5.5 复用性

设计完的ETL转换应该可以被复用，这也是ETL工具的一个不可或缺的特征。复制和粘贴已存在的转换步骤是最常见的一种复用，但这还不是真正意义上的复用。复用一词是指定义了一个转换或步骤，从其他地方可以调用这些转换或步骤。Kettle 里有一个映射（子转换）步骤，可以完成转换的复用，该步骤可以将一个转换作为其他转换的子转换。另外转换还可以在多个作业里多次使用，同样作业也可以作为其他作业的子作业。

1.5.6 扩展性

世界上没有一款ETL工具为你提供了所有能想象到的数据转换的能力，Kettle 也是如此。这就意味着必须要有扩展功能的方法。几乎所有的ETL工具都提供了脚本，以编程的方式来解决工具本身不能解决的问题。另外有少数几款ETL工具可以通过API或其他方式来为工具增加组件。第三种方法是使用脚本语言写函数，函数可以被其他转换或脚本调用。

Kettle提供了上述所有功能。Java脚本步骤可以用来开发Java脚本，把这个脚本保存为一个转换，再通过映射（子转换）步骤，又可以变为一个标准的可以复用的函数。实际上，并不限于脚本，每个转换都可以通过这种映射（子转换）方式来复用，如同创建了一个组件。Kettle 在设计上就是可扩展的，它提供了一个插件平台。这种插件架构允许第三方为Kettle平台开发插件。本书也将介绍几个Kettle 插件，但你也要知道Kettle 里的所有组件，即使是默认提供的组件，实际上也都是插件。内置的第三方插件和Pentaho插件的唯一区别是技术支持。如果你买了一个第三方插件（例如一个SugarCRM的连接），技术支持由第三方提供，而不是由Pentaho 提供。

1.5.7 数据转换

ETL项目很大一部分工作都是在做数据转换。在输入和输出之间，数据要经过检验、连接、分割、合并、转置、排序、合并、克隆、排重、过滤、删除、替换或者其他操作。在不同机构、项目、解决方案里，数据转换的需求都大不相同，所以很难说清一个ETL工具最少应该提供哪些转换功能。但是常用的ETL工具（包括Kettle）都提供了下面一些最基本的整合功能：

- 缓慢变更维度
- 查询值
- 行列转换
- 条件分割
- 排序、合并、连接
- 聚集

各ETL工具之间唯一的不同就是转换如何定义。例如，一些工具只用一个步骤就提供了标准的SCD（缓慢变更维度）转换，而其他工具则是通过向导生成需要的转换。当然Kettle 也不能满足所有转换的需求。如Kettle里没有层次扁平化组件。这里的层次指表的自引用，如雇员表，每行记录都有一个雇员ID和一个经理ID。经理ID指向了雇员表里的另一个是经理的雇员。Oracle 有一个标准的“connect by prior” 功能来解决这个问题，其他一些ETL工具也有类似功能，在Kettle里可能要通过写脚本来处理这个问题。（译者注：通过“闭合生成器”（Closure

Generator) 步骤和排序、分组等步骤的组合也能解决自引用表的相关问题。)

1.5.8 测试和调试

测试和调试的重要性不言而喻。即使我们设计ETL转换并不需要Java或C++等开发语言, 但我们的设计过程和直接用开发语言写程序也很相似。也就是说在写程序时用到的一些步骤或过程同样也适用于ETL设计, 测试也是ETL设计的一部分。为了完成测试工作, 我们通常要假设下面几种场景, 并要给出相应的解决方法:

- 如果ETL过程没有按时完成数据转换的任务怎么办?
- 如果转换过程异常终止怎么办?
- 目标是非空列的数据抽取到的数据为空怎么办?
- 转换后的行数和抽取到的数据行数不一致怎么办(数据丢失)?
- 转换后计算的数值和另一个系统的数值不一致怎么办(逻辑错误)?

这里要讲的就是失败处理方法。ETL的过程中不要把事情想象得太顺利, 要考虑到在某个转换中某个节点失败的可能性。测试通常分为黑盒测试(也叫功能测试)和白盒测试。对于前者, ETL转换就被认为是一个黑盒子, 测试者并不了解黑盒子内的功能, 只知道输入和期望的输出。白盒测试(也叫结构测试)要求测试者知道转换内部的工作机制并依此设计测试用例来检查特定的转换是否有特定的结果。两种测试方法都有它的优缺点, 在第11章中将讲述这两种测试方法。

调试实际是白盒测试中的一部分, 通过调试可以让开发者或测试者一步一步地运行一个转换, 并找出问题的所在。并非所有ETL工具都提供了单步逐行调试功能。Kettle为作业和转换都提供了这种特性, 在第11章将详细讲述。

1.5.9 血统和影响分析

任何ETL工具都应该有一个重要的功能: 读取转换的元数据, 提取由不同转换构成的数据流的信息。血统分析和影响分析是基于元数据的两个相关的特性。血统是一种回溯性的机制, 它可以查看到数据的来源。例如, “价格”和“数量”字段作为输入字段, 在转换中根据这两个字段计算出“收入”字段。即使在后面的处理流程里过滤了“价格”和“数量”字段, 血统分析功能也能分析出“收入”字段是基于“价格”和“数量”字段的。

影响分析是基于元数据的另一种分析方法: 该方法可以分析源数据字段对随后的转换以及目标表的影响, 我们将在第14章中详细讲述这一主题。

1.5.10 日志和审计

数据仓库的目的就是要提供一个准确的信息源, 因此数据仓库里的数据应该是可靠的、可信任的。为了保证这种可靠性, 同时保证可以记录下所有的数据转换操作, ETL工具应该提供日志和审计功能。日志可以记录下在转换过程中执行了哪些步骤, 包括每个步骤开始和结束时间戳。审计可以追踪到对数据做的所有操作, 包括读行数、转换行数、写行数。在这方面 Kettle 在 ETL工具市场处于领先地位, 我们将在第12章和第14章讲述这两个功能。

1.6 小结

本章介绍了ETL和它的历史，并解释了数据整合的必要性。简要介绍了Kettle的基本构成模块，使读者对Kettle有了大致的了解。我们还解释了ETL、ELT、EII之间的相同点和不同点以及各自的优点。我们还列举了开发ETL解决方案中可能面临的挑战，包括：

- 获得业务部门的支持。
- 选择一个好的工作方法。
- 设计ETL解决方案。
- 数据获取和电子表格问题。
- 使用数据分析和检验处理数据质量问题。

最后我们说明了ETL工具应该具备的通用特性，以及 Kettle如何满足这些特性：

- 连接
- 平台独立和规模扩展性
- 设计灵活和组件复用
- 可扩展性
- 数据转换
- 测试和调试
- 血统和影响分析
- 日志和审计

本书的其余章节将详细展开本章中介绍的这些主题。

CHAPTER

2

第2章 Kettle基本概念

本章主要讲述Kettle的基本概念，我们需要了解 Kettle 工具本身的一些设计原则，以及Kettle里的不同功能模块。首先讲述如何通过转换，以数据行的形式来处理数据，然后解释如何使用作业以工作流的形式将转换连接起来。

本章要讲述如下的Kettle概念：

- 数据库连接
- 工具和常用程序
- 资源库
- 虚拟文件系统
- 参数和变量
- 可视化编程

2.1 设计原则

Kettle工具在设计初始就考虑到了一些设计原则。这些原则也借鉴了以前使用过的其他一些ETL工具积累下的经验和教训。先总结一下以前的经验，看我们能从中获得哪些有益的经验。

- **易于开发：**作为数据仓库和ETL开发者，你只想把时间用在创建BI 解决方案上。任何用于软件安装、配置的时间都是一种浪费。例如，为了创建数据库连接，很多和Kettle 类似的Java工具都要求用户手工输入数据库驱动类名和JDBC URL 连接串。尽管用户通过互联网都能搜索到这些信息，但这明显把用户的注意力转移到了技术方面而非业务方面。Kettle 尽量避免这类问题的发生。
- **避免自定义开发：**一般来说，ETL工具要使简单的事情更简单，使复杂的事情成为可能。ETL工具提供了标准化的构建组件来实现ETL开发人员不断重复的需求。当然可以通

过手工写Java代码或Java脚本来实现一些功能。但增加的每一行代码都给项目增加了复杂度和维护成本。所以尽量避免手工开发，尽量使用已提供组件的各种组合来完成任务。

- **所有功能都通过用户界面完成：**对于这一黄金准则也有很少的几个例外（如kettle.properties和shared.xml文件就是两个例外，不能通过界面，要手工修改配置文件），如果不直接把所有功能通过界面的方式提供给用户，实际上就是在浪费开发人员的时间，也是在浪费用户的时间。专家级的ETL用户还要去学习隐藏在界面以外的一些特性。在Kettle里，ETL元数据可以通过XML格式表现，或通过资源库，或通过使用Java API。无论ETL元数据是以哪种形式提供，都可以百分之百通过图形用户界面来编辑。

- **没有命名限制：**ETL转换里有各种各样的名称，如数据库连接、转换、步骤、数据字段、作业等都要有一个名称。如果还要在命名时考虑到一些限制（如长度、选择的字符），就会给工作带来一定麻烦。ETL工具需要足够的智能化来处理ETL开发人员设置的各种名称。最终ETL解决方案应该可以尽可能地自描述，这样可以部分减少文档的需求，减少项目维护成本。

- **透明：**如果有ETL工具需要你了解转换中某一部分工作是如何完成的，那么这个ETL工具就是不透明的。当然，如果想自己实现ETL工具里某一个同样的功能，你就要确切地知道这一部分功能是如何完成的。不过允许用户看到ETL过程中各部分的运行状态也是很重要的，这样可以加快开发速度、降低维护成本。

ETL工作流程中的不同部分不能互相影响，它们应该只是以指定的顺序传递数据。这种数据隔离的原则也在很大程度上影响了透明性，那些使用非数据隔离的ETL工具的用户会感受到透明的益处。

- **灵活的数据通道：**对ETL开发者来说，创造性极端重要，创造性不但让你享受到工作的乐趣，而且能让你以最快的方式开发出ETL方案。Kettle从设计初始就在数据的发送、接收方式上尽可能灵活。Kettle可以在文本文件、关系数据库等不同目标之间复制和分发数据，从不同数据源合并数据也是内核引擎的一部分，也同样很简单。
- **只映射需要映射的字段：**在一些ETL工具里经常可以看到数百行的输入和输出映射，对于维护人员来说这是一个噩梦。在ETL开发过程中，字段要经常变动，这样的大量映射也会增加维护成本。

Kettle的一个重要核心原则就是在ETL流程中所有未指定的字段都自动被传递到下一个组件。这个原则极大减少了维护成本。也就是说输入中的字段会自动出现在输出中，除非中间过程特别设置了终止某个字段的传递。

2.2 Kettle设计模块

通常每个ETL工具都用不同的名字来区分不同的组成部分，名字就说明了这一组成部分的功能，或可以使用这一部分得到什么结果，Kettle也不例外。本节解释了一些Kettle特定的功能名称。通过阅读本章可以了解如何在转换里逐行处理转换，如何在作业里处理工作流，你也能学习到诸如数据类型和数据转换等细节内容。

2.2.1 转换

转换（transformation）是ETL解决方案中最主要的部分，它处理抽取、转换、加载各阶段各

种对数据行的操作。转换包括一个或多个步骤（step），如读取文件、过滤输出行、数据清洗或将数据加载到数据库。

转换里的步骤通过跳（hop）来连接，跳定义了一个单向通道，允许数据从一个步骤向另一个步骤流动。在Kettle 里，数据的单位是行，数据流就是数据行从一个步骤到另一个步骤的移动。数据流的另一个同义词就是记录流。

图2-1显示了一个转换例子，该转换从数据库表读取数据并写入到文本文件。

除了步骤和跳，转换还包括了注释（note），注释是一个小的文本框，可以放在转换流程图的任何位置。注释的主要目的是使转换文档化。

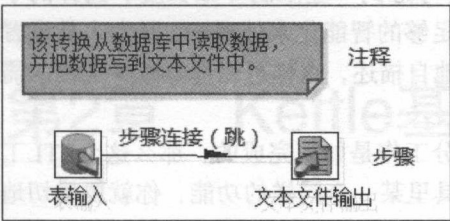


图2-1 一个简单转换的例子

步骤

步骤是转换里的基本组成部分。它以图标的方式图形化地展现，图2-1显示了两个步骤，“表输入”和“文本文件输出”。一个步骤有如下几个关键特性：

- 步骤需要有一个名字，这个名字在转换范围内唯一。
- 每个步骤都会读、写数据行（唯一例外是“生成记录”步骤，该步骤只写数据）。
- 步骤将数据写到与之相连的一个或多个输出跳（outgoing hops），再传送到跳的另一端的步骤。对另一端步骤来说这个跳就是一个输入跳（incoming hops），步骤通过输入跳接收数据。
- 大多数的步骤都可以有多个输出跳。一个步骤的数据发送可以被设置为轮流发送和复制发送。轮流发送是将数据行依次发给每一个输出跳（这种方式也称为round robin），复制发送是将全部数据行发送给所有输出跳。
- 在运行转换时，一个线程运行一个步骤和步骤的多份拷贝，所有步骤的线程几乎同时运行，数据行连续地流过步骤之间的跳。

除了上面这些标准的功能，每个步骤都有明显的功能区别，这可以通过步骤类型体现。如图2-1中的“表输入”步骤就是向关系型数据库的表发出一个SQL查询，并将得到的数据行写到它的输出跳，另一方面“文本文件输出”步骤从它的输入跳读取数据行，并将数据行写到文本文件。

转换的跳

跳（hop）就是步骤之间带箭头的连线，跳定义了步骤之间的数据通路。跳实际上是两个步骤之间的被称为行集（row set）的数据行缓存（行集的大小可以在转换的设置里定义）。当行集

满了，向行集写数据的步骤将停止写入，直到行集里又有了空间。当行集空了，从行集读取数据的步骤停止读取，直到行集里又有可读的数据行。

注意：当创建新跳的时候，需要记住跳在转换里不能循环。因为在转换里每个步骤都依赖前一个步骤获取字段值。

并行

跳的这种基于行集缓存的规则允许每个步骤都由一个独立的线程运行，这样并发程度最高。这一规则也允许数据以最小消耗内存的数据流的方式来处理。在数据仓库里，我们经常要处理大量数据，所以这种并发低耗内存的方式也是ETL工具的核心需求。

对于 Kettle，不可能定义一个执行顺序，不可能也没有必要确定一个起点和终点。因为所有步骤都以并发方式执行：当转换启动后，所有步骤都同时启动，从它们的输入跳中读取数据，并把处理过的数据写到输出跳，直到输入跳里不再有数据，就中止步骤的运行。当所有的步骤都中止了，整个转换就中止了。也就是说，从功能的角度来看，转换也有明确的起点和终点。例如，图2-1 里显示的转换起点就是“表输入”步骤（因为这个步骤生成数据行），终点就是“文本文件输出”步骤（因为这个步骤将数据写到文件，而且后面不再有其他节点）。

上面说的如何定义转换的起点和终点看上去有点自相矛盾或截然相反。实际上，并没有这么复杂，只是因为看问题的角度不同。你能想象到数据沿着转换里的步骤移动，而形成一条从头到尾的数据通路。而另一方面，转换里的步骤几乎是同时启动的，所以不可能判断出哪个步骤是第一个启动的步骤。

如果想要一个任务沿着指定的顺序执行，那么就要使用本章后面讲的“作业”了。

数据行

数据以数据行的形式沿着步骤移动。一个数据行是零到多个字段的集合，字段包括下面几种数据类型。

- String：字符类型数据。
- Number：双精度浮点数。
- Integer：带符号长整型（64位）。
- BigInteger：任意精度数值。
- Date：带毫秒精度的日期时间值。
- Boolean：取值为true和false的布尔值。
- Binary：二进制字段可以包括图形、声音、视频及其他类型的二进制数据。

每个步骤在输出数据行时都有对字段的描述，这种描述就是数据行的元数据，通常包括下面一些信息。

- 名称：行里的字段名应该是唯一的。
- 数据类型：字段的数据类型。
- 长度：字符串的长度或BigInteger类型的长度。
- 精度：BigInteger数据类型的十进制精度。
- 掩码：数据显示的格式（转换掩码）。如果要把数值型（Number、Integer、BigInteger）

或日期类型转换成字符串类型就需要用到掩码。例如在图形界面中预览数值型、日期型数据，或者把这些数据保存成文本或XML格式就需要用到这种转换。

- 小数点：十进制数据的小数点格式。不同文化背景下小数点符号是不同的，一般是点 (.) 或逗号 (,)。
- 分组符号：数值类型数据的分组符号，不同文化背景下数字里的分组符号也是不同的，一般是逗号 (,) 或点 (.) 或单引号 (')。（译者注：分组符号是数字里分割符号，便于阅读，如123,456,789。）
- 初始步骤：Kettle 在元数据里还记录了字段是由哪个步骤创建的。可以让你快速定位字段是由转换里的哪个步骤最后一次修改或创建。

当设计转换时有几个数据类型的规则需要注意：

- 行级里的所有行都应该有同样的数据结构。就是说：当从多个步骤向一个步骤里写数据时，多个步骤输出的数据行应该有相同的结构，即字段相同、字段数据类型相同、字段顺序相同。
- 字段元数据不会在转换中发生变化。就是说：字符串不会自动截去长度以适应指定的长度，浮点数也不会自动取整以适应指定的精度。这些功能必须通过一些指定的步骤来完成。
- 默认情况下，空字符串 ("") 被认为与NULL 相等。

注意：空字符串与NULL是否相等，可以通过一个参数KETTLE_EMPTY_STRING_DIFFERS_FROM_NULL来设置。更多详细内容见附录C。

数据转换

既可以显式地转换数据类型，如在“字段选择”步骤中直接选择要转换的数据类型，也可以隐式地转换数据类型，如将数值类型数据写入数据库中的VARCHAR类型字段。这两种形式的数据转换实际是完全一样的，都是使用了数据和对数据的描述。

Date和String的转换

Kettle 内部的Date 类型里包括了足够的信息，可以用这些信息来表现任何毫秒精度的日期、时间值。如果要在 String和Date 类型之间转换，唯一要指定的就是日期格式掩码。关于日期和时间的掩码格式可以参考Java API文档的“Date and Time Patterns”部分，<http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>。这个文档里列出了所有可以用于编码的字母，不用于编码目的的字母都应该包含在单引号里。

例如，表2-1显示了一个例子日期2009年12月6日21点6分54秒321毫秒的几个字符串日期编码的例子。

表2-1 日期转换例子

| 转换掩码 (格式) | 结果 |
|-----------------------------|-------------------------|
| yyyy/MM/dd 'T' HH:mm:ss.SSS | 2009/12/06T21:06:54.321 |
| h:mm a | 9:06 PM |
| HH:mm:ss | 21:06:54 |
| M-d-yy | 12-6-09 |

Numeric和String的转换

Numeric数据（包括Number、Integer、BigNumber）和String 类型之间的转换用到了下面几种字段的元数据。

- 转换掩码
- 小数点符号
- 分组符号
- 货币符号

这些转换掩码只是决定了一个文本格式的字符串如何转换为一个数值，而与数值本身的实际精度和舍入无关。Java API中定义了所有可用的掩码符号和格式规则，参考 <http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html>。

表2-2 显示了几个常用的例子。

表2-2 几个数值转换掩码的例子

| 值 | 转换掩码 | 小数点符号 | 分组符号 | 结果 |
|-----------|---------------|-------|------|---------------|
| 1234.5678 | #,###.## | . | , | 1,234.57 |
| 1234.5678 | 000,000.00000 | , | . | 001.234,56780 |
| -1.9 | #.00; -#.00 | . | , | -1.9 |
| 1.9 | #.00; -#.00 | . | , | 1.9 |
| 12 | 00000;-00000 | | | 00012 |

其他转换

表2-3 提供了其他几种数据类型转换的列表。

表2-3 其他数据类型转换

| 从 | 到 | 描述 |
|---------|---------|--|
| Boolean | String | 转换为Y或N，如果设置长度大于等于3，转换成 true或false |
| String | Boolean | 字符串Y、True、Yes、1 都转换为true，其他字符串转换为false（不区分大小写） |
| Integer | Date | 整型和日期型之间转换时，整型就是从1970-01-01 00:00:00 GMT 开始计算的 |
| Date | Integer | 毫秒值。例如2010-0912可以转换成 1284112800000，反之亦然 |

2.2.2 作业

大多数ETL项目都需要完成各种各样的维护工作。例如，当运行中发生错误，要做哪些操作；如何传送文件；验证数据库表是否存在，等等。而且这些操作要按照一定顺序完成。因为转换以并行方式执行，就需要一个可以串行执行的作业来处理这些操作。

一个作业包括一个或多个作业项，这些作业项以某种顺序来执行。作业执行顺序由作业项之间的跳（job hop）和每个作业项的执行结果来决定。图2-2 显示了一个典型的加载数据仓库的作业。

如同转换，作业里也可以包括注释。

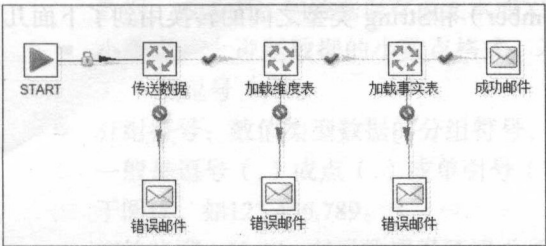


图2-2 一个典型的加载数据仓库的作业

作业项

作业项是作业的基本构成部分。如同转换的步骤，作业项也可以使用图标的方式图形化展示。但是，如果你再仔细观察，还是会发现作业项有一些地方不同于步骤：

- 新步骤的名字应该是唯一的，但作业项可以有影子拷贝。这样可以把一个作业项放在多个不同的位置。这些影子拷贝里的信息都是相同的，编辑了一份拷贝，其他拷贝也会随之修改。
- 在作业项之间可以传递一个结果对象（result object）。这个结果对象里包含了数据行，它们不是以流的方式来传递的。而是等一个作业项执行完了，再传递给下一个作业项。
- 默认情况下，所有的作业项都是以串行方式执行的，只是在特殊的情况下，以并行方式执行。

因为作业顺序执行作业项，所以必须定义一个起点。有一个叫“开始”的作业项就定义了这个起点。一个作业只能定义一个开始作业项。

作业跳

作业的跳是作业项之间的连接线，它定义了作业的执行路径。作业里每个作业项的不同运行结果决定了作业的不同执行路径。对作业项的运行结果的判断如下。

- 无条件执行：不论上一个作业项执行成功还是失败，下一个作业项都会执行。这是一种黑色的连接线，上面有一个锁的图标，如图2-2所示。
- 当运行结果为真时执行：当上一个作业项的执行结果为真时，执行下一个作业项。通常需要在需要无错误执行的情况下使用。这是一种绿色的连接线，上面有一个对钩号的图标，如图2-2所示。
- 当运行结果为假时执行：当上一个作业项的执行结果为假或没有成功执行时，执行下一个作业项。这是一种红色的连接线，上面有一个红色的停止图标。

在作业项连接（跳）的右键菜单上和跳的小图标的选项里都可以设置上面这几种判断方式。

多路径和回溯

Kettle 使用一种回溯算法来执行作业里的所有作业项，而且作业项的运行结果（真或假）也决定执行路径。回溯算法就是：假设执行到了图里的一条路径的某个节点时，要依次执行这个

节点的所有子路径，直到没有再可以执行的子路径，就返回该节点的上一节点，再反复这个过程。

例子如图2-3所示。

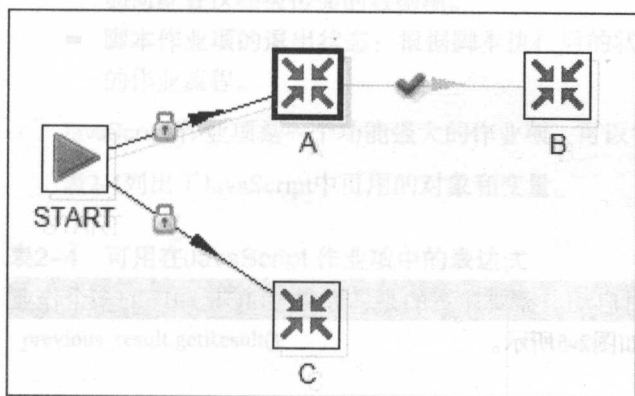


图2-3 使用回溯算法串行执行多个路径

图2-3里的A、B、C三个作业项的执行顺序如下。

- 首先“开始”作业项搜索所有下一个节点作业项，找到了“A”和“C”。
- 执行“A”。
- 搜索“A”后面的作业项，发现了“B”。
- 执行“B”。
- 搜索“B”后面的作业项，没有找到任何作业项。
- 回到“A”，也没有发现其他作业项。
- 回到Start，发现另一个要执行的作业项“C”。
- 执行“C”。
- 搜索“C”后面的作业项，没有找到任何作业项。
- 回到Start，没有找到任何作业项。
- 作业结束。

因为没有定义执行顺序，所以上面例子的执行顺序除了ABC，还可以有CAB。

这种回溯算法有两个重要特征：

- 因为作业是可以嵌套的，除了作业项有运行结果，作业也需要一个运行结果，因为一个作业可以是另一个作业的作业项。一个作业的运行结果，来自于它最后一个执行的作业项。上面的例子里作业的执行顺序可以是ABC，也可以是CAB，所以不能保证作业项C的结果就是作业的结果。
- 当你在作业里创建了一个循环（作业里允许循环），一个作业项就会被执行多次，作业项的多次运行结果会保存在内存里，便于以后使用。第15章将详细讲解这一主题。

并行执行

有时候需要将作业项并行执行。这种并行执行也是可以的。一个作业项可以并发的方式执行它后面的所有作业项，如图2-4所示。

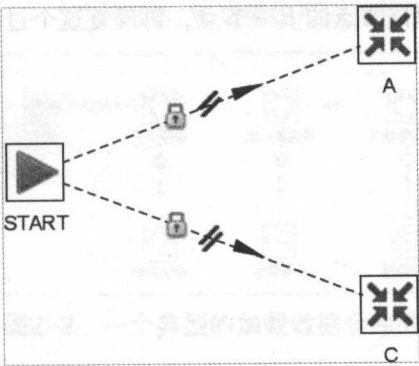


图2-4 并行执行的作业项

在这个例子里，作业项A和C几乎同时启动。需要注意的是，如果A和C是顺序的多个作业项，那么这两组作业项也是并行执行的，如图2-5所示。

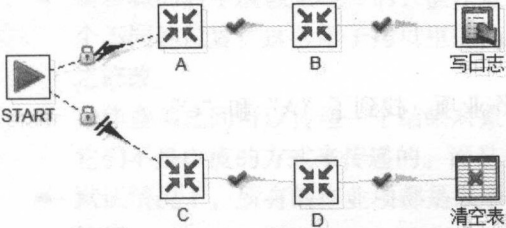


图2-5 两组同时执行的作业项

在这个例子里，作业项[A, B, Log1]和[C, D, Cleanup Tables]是在两个线程里并行执行的。通常设计者也是希望以这样的方式执行。但有时候，设计者希望一部分作业项并行执行，然后再串行执行其他作业项。这就需要把要并行的作业项放到一个新的作业里，然后作为另一个作业的作业项，如图2-6所示。



图2-6 并行加载作业作为另一个作业的作业项

作业项结果

作业执行结果不仅决定了作业的执行路径，而且还向下一个作业项传递了一个结果对象。结果对象包括了下面一些信息。

- 一组数据行：在转换里使用“复制行到结果”步骤可以设置这组数据行。与之对应，使用“从结果获取行”步骤可以获取这组数据行。在一些作业项里，如“Shell脚本”、“转换”、“作业”的设置里有一个选项可以循环这组数据行，这样可以通过参数化来控制转换和作业。
- 一组文件名：在作业项的执行过程中可以获得一些文件名（通过步骤的选项：“添加到结果文件”。——译者注）。这组文件名是所有与作业项发生过交互的文件的名称。例如，一个转换读取和处理了10个XML文件，这些文件名就会保留在结果对象里。使用转

换里的“从结果获取文件”步骤可以获取到这些文件名，除了文件名还能获取到文件类型。“一般”类型是指所有的输入输出文件，“日志”类型是指Kettle日志文件。

- 读、写、输入、输出、更新、删除、拒绝的行数和转换里的错误数：第14章详细说明了如何配置这些被传递的数据项。
- 脚本作业项的退出状态：根据脚本执行后的状态码，判断脚本的运行状态，再执行不同的作业流程。

JavaScript 作业项是一个功能强大的作业项，可以实现更高级的流程处理功能。

表2-4列出了JavaScript中可用的对象和变量。

表2-4 可用在JavaScript 作业项中的表达式

| 表达式 | 数据类型 | 含义 |
|--|---------|---------------------------------------|
| previous_result.getResult() | Boolean | true: 前一个作业项成功执行 false: 前一个作业项执行错误 |
| previous_result.getExitStatus()或exit_status | Int | 前一个脚本作业项的退出状态码 |
| previous_result.getEntryNr()或nr | Int | 已执行的作业项的个数：每执行一个作业项增加一个 |
| previous_result.getNrErrors()或errors | long | 错误个数 |
| previous_result.getNrLinesInput()或lines_input | long | 从文件或数据库里读到的行数，输入行数 |
| previous_result.getNrLinesOutput()或lines_output | long | 写到文件或数据库里的行数，输出行数 |
| previous_result.getNrLinesRead()或lines_read | long | 从上一个步骤里读到的行数，读行数 |
| previous_result.getNrLinesUpdated()或lines_updated | long | 对文件或数据库更新的行数，更新行数 |
| previous_result.getNrLinesWritten()或lines_written | long | 写到下一个步骤里的行数，写行数 |
| previous_result.getNrLinesDeleted()或lines_deleted | long | 删除的行数 |
| previous_result.getNrLinesRejected()或lines_rejected | long | 发生错误被拒绝，并通过错误处理传给下一个步骤的行数 |
| previous_result.getRows() | List | 结果行数 |
| previous_result.getResultFileList() | List | 前面作业项里用到的所有文件列表 |
| previous_result.getNrFileRetrieved()或files_retrieved | Int | 从FTP、SFTP等处获得的文件 |

在JavaScript作业项里，可以设置一些条件，这些条件的结果，可以决定最终执行哪条作业路径。

例如，可以统计在一个转换里被拒绝的总行数。如果总数超过50，可以认为转换失败。脚本如下：

```
Lines_rejected <= 50
```


2.2.3 转换或作业的元数据

转换和作业是Kettle的核心组成部分。以前曾经讨论过，它们可以用XML格式来表示，可以保存在资源库里，可以用Java API的形式来表示。它们的这些表示方式，都依赖于下面的这些元数据。

- **名字：**转换或作业的名字，尽管名字不是必要的，但应该使用名字，不论是在一个ETL工程内还是在多个 ETL工程内，都应尽可能使用唯一的名字。这样在远程执行时或多个 ETL工程共用一个资源库时都会有帮助。
- **文件名：**转换或作业所在的文件名或URL。只有当转换或作业是以XML文件的形式存储时，才需要设置这个属性。当从资源库加载时，不必设置这个属性。
- **目录：**这个目录，是指在Kettle资源库里的目录，当转换或作业保存在资源库里时设置。当保存为 XML文件时，不用设置。
- **描述：**这是一个可选属性，用来设置作业或转换的简短的描述信息。如果使用了资源库，这个描述属性会出现在资源库浏览窗口的文件列表中。
- **扩展描述：**也是一个可选属性，用来设置作业或转换的详细描述信息。

2.2.4 数据库连接

Kettle 里的转换和作业使用数据库连接来连接到关系型数据库。Kettle数据库连接实际是数据库连接的描述：也就是建立实际连接需要的参数。实际连接只是在运行时才建立，定义一个Kettle的数据库连接并不真正打开一个到数据库的连接。

不幸的是各个数据库的行为都不是完全相同的。所以，在图2-7的数据库连接窗口里有很多种数据库，而且数据库的种类还在增多。

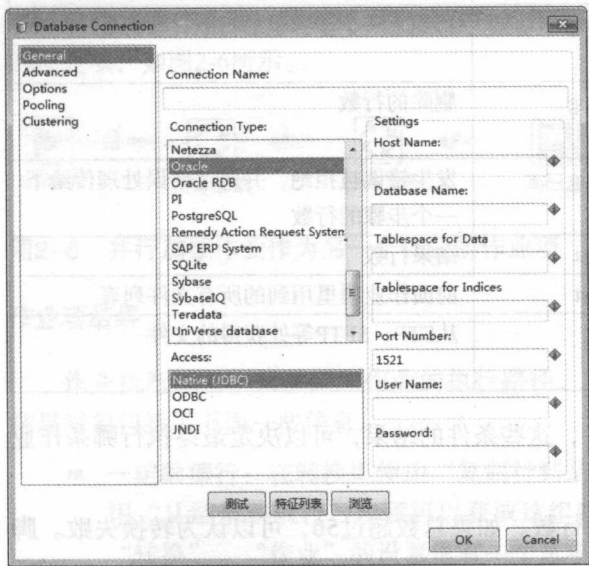


图2-7 数据库连接窗口

在数据库连接窗口中主要设置下面3个选项。

- **连接名称：**设定一个在作业或转换范围内唯一的名称。

- **连接类型**：从数据库列表中选择要连接的数据库类型。根据选中数据库的类型不同，要设置的访问方式和连接参数设置也不同。某些Kettle步骤或作业项生成SQL语句时使用的方言也不同。
- **访问方式**：在列表里可以选择可用的访问方式，一般都使用JDBC连接。不过也可以使用ODBC数据源、JNDI数据源、Oracle的OCI连接（使用Oracle命名服务）。

根据选择的数据库不同，右侧面板的连接参数设置也不同。例如，在图2-7里只有Oracle数据库可以设置表空间选项。

一般常用的连接参数如下。

- **主机名**：数据库服务器的主机名或IP地址。
- **数据库名**：要访问的数据库名。
- **端口号**：默认是选中的数据库服务器的默认端口号。
- **用户名和密码**：数据库服务器的用户名和密码。

特殊选项

对大多数用户来说，使用数据库连接窗口的“一般”标签就足够了。但偶尔也可能需要设置对话框里的“高级”标签的内容。

- **支持Boolean数据类型**：对Boolean（bit）数据类型，大多数数据库的处理方式都不相同。即使同一个数据库的不同版本也有不同。许多数据库根本不支持Boolean类型。所以默认情况下，Kettle使用一个字符的字段（char(1)）的不同值（Y或N）来代替Boolean字段。如果选中了这个选项，Kettle就会为支持Boolean类型的数据库生成正确的SQL方言。
- **双引号分割标识符**：强迫SQL语句里的所有标识符（列名、表名）加双引号，一般用于区分大小写的数据库，或者你怀疑Kettle里定义的关键字列表和实际数据库不一致。
- **强制转为小写**：将所有标识符（表名和列名）转为小写。
- **强制转为大写**：将所有标识符（表名和列名）转为大写。
- **默认模式名**：当不明确指定模式名（有些数据库里叫目录）时，默认的模式名。
- **连接后要执行的SQL语句**：一般用于建立连接后，修改某些参数，如Session级的变量或调试信息等。

除了这些高级选项，在连接对话框的“选项”标签下，还可以设置数据库特定的参数，如一些连接参数。为便于使用，对于某些数据库（如MySQL），Kettle提供了一些默认的连接参数和值。各个数据库详细的参数列表，参考数据库JDBC驱动手册。有几种数据库类型，Kettle还提供了连接参数的帮助文档，通过单击“选项”标签中的“帮助”按钮可以打开对应数据库的帮助页面。

最后，还可以选择Apache的通用数据库连接池的选项。如果你运行了很多小的转换或作业，这些转换或作业里又定义了生命期短的数据库连接，连接池选项就显得有意义了。连接池选项不会限制并发数据库连接的数量，该选项在第15章将详细说明。

关系型数据库的力量

关系型数据库是一种高级的软件，它在数据的连接、合并、排序等方面有着突出的优势。

和基于流的数据处理引擎，如 Kettle，相比，它有一大优点：数据库使用的数据都存储在磁盘中。当关系型数据库进行连接或排序操作时，只要使用这些数据的引用即可，而不用把这些数据加载到内存里，这就体现出明显的性能方面的优势。但缺点也是很明显的，把数据加载到关系型数据库里也会产生性能的瓶颈。

对ETL开发者而言，要尽可能利用数据库自身的性能优势，来完成连接或排序这样的操作。如果不能在数据库里进行连接这样的操作，如数据的来源不同，也应该先在数据库里排序，以便在ETL里做连接操作。

连接和事务

数据库连接只在执行作业或转换时使用。在作业里，每一个作业项都打开和关闭一个独立的数据库连接。转换也是如此。但是因为转换里的步骤是并行执行的，每个步骤都打开一个独立的数据库连接并开始一个事务。尽管这样在很多情况下会提高性能，但当不同步骤更新同一个表时，也会带来锁和参照完整性问题。

为解决打开多个连接而产生的问题，Kettle可以在一个事务中完成转换。转换设置对话框的选项“转换放在数据库事务中”，可以完成此功能。当选中了这个选项，所有步骤里的数据库连接都使用同一个数据库连接。只有所有步骤都正确，转换正确执行，才提交事务，否则回滚事务。

数据库集群

当一个大数据库不再能满足需求时，你就会考虑用很多小的数据库来处理数据。通常可以使用数据库分区或称数据库分片技术来分散数据加载。这种方法可以将一个大数据集分为几个数据组成为分区（或分片），每个分区都保存在独立的数据库实例里。这种方法的优点显而易见，可以大幅减少每个表或每个数据库实例的行数。所有分片的组合就是数据库集群。

一般采用标识符计算余数的方法来决定分片的数据保存到哪个数据库实例里。除此之外，Kettle 里还有其他几种分区方法（见第16章）。

上面说的分区计算方法得到的分区标识是一组0到“分区数-1”之间的数字，可以在数据库连接对话框的“集群”标签下设置分区数。例如，你定义了五个数据库连接作为集群里的五个数据分片。可以在“表输入”步骤里执行一个查询，这个查询就以分区的方式执行，如图2-8所示。

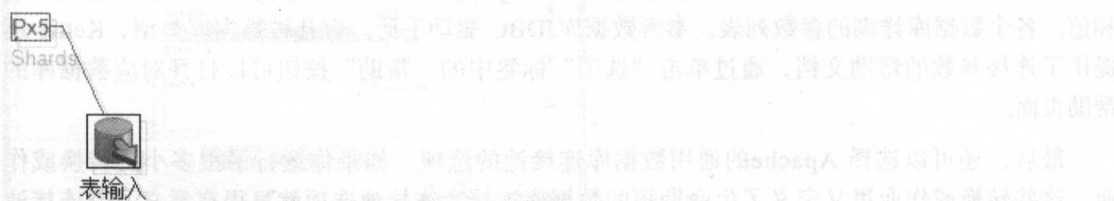


图2-8 数据库集群的表输入

在上图中，同样的一个查询会被执行五遍，每个数据分区执行一遍。在Kettle 里，所有使用数据库连接的步骤都可以使用分区的特性。例如，表输出步骤在分区模式下会把不同的数据行输出到不同的数据分区（片）。

2.2.5 工具

Kettle里有不同工具，用于ETL的不同阶段。主要工具如下。

- **Spoon**: 图形界面工具，快速设计和维护复杂的ETL workflow。
- **Kitchen**: 运行作业的命令行工具。
- **Pan**: 运行转换的命令行工具。
- **Carte**: 轻量级的（大概1MB）Web服务器，用来远程执行转换或作业。一个运行有Carte进程的机器可以作为从服务器，从服务器是Kettle集群的一部分。

第3章将详细介绍这些工具。

2.2.6 资源库

当你的ETL项目规模比较大，有很多ETL开发人员在一起工作，开发人员之间的合作就显得很重要。Kettle以插件的方式灵活地定义不同种类的资源库，但不论是哪种资源库，它们的基本要素是相同的：这些资源库都使用相同的用户界面、存储相同的元数据。目前有3种常见资源库：数据库资源库、Pentaho资源库和文件资源库。

- **数据库资源库**: 数据库资源库是把所有的ETL信息保存在关系型数据库中，这种资源库比较容易创建，只要新建一个数据库连接即可。可以使用“数据库资源库”对话框来创建资源库里的表和索引。
- **Pentaho资源库**: Pentaho资源库是一个插件，在Kettle的企业版中有这个插件。这种资源库实际是一个内容管理系统（CMS），它具备一个理想的资源库的所有特性，包括版本控制和依赖完整性检查。
- **文件资源库**: 文件资源库是在一个文件目录下定义一个资源库。因为Kettle使用的是虚拟文件系统（Apache VFS——译者注），所以这里的文件目录是一个广泛的概念，包括了zip文件、Web服务、FTP服务等。

无论哪种资源库都应该具有下面的特性。

- **中央存储**: 在一个中心位置存储所有的转换和作业。ETL用户可以访问到工程的最新视图。
- **文件加锁**: 防止多个用户同时修改。
- **修订管理**: 一个理想的资源库可以存储一个转换或作业的所有历史版本，以便将来参考。你可以打开历史版本，并查看变更日志。详见第13章。
- **依赖完整性检查**: 检查资源库转换或作业之间的相互依赖关系，可以确保资源库里没有丢失任何链接，没有丢失任何转换、作业或数据库连接。
- **安全性**: 安全性可以防止未授权的用户修改或执行ETL作业。
- **引用**: 重新组织转换、作业，或简单重新命名，都是ETL开发人员的常见工作。要做好这些工作，需要完整的转换或作业的引用。

2.2.7 虚拟文件系统

灵活而统一的文件处理方式对ETL工具来说非常重要。所以Kettle支持URL形式的文件名，Kettle使用Apache的通用VFS作为文件处理接口，替你解决各种文件处理方面的复杂情况。例

如，使用Apache VFS可以选中.zip压缩包内的多个文件，和在一个本地目录下选择多个文件一样方便。关于VFS的更多信息，请访问 <http://commons.apache.org/vfs/>。

表2-5是VFS的一些典型的例子。

表2-5 VFS文件规范的例子

| 文件名例子 | 描述 |
|---|--|
| 文件名: /data/input/customers.dat | 这是最典型的定义文件的方式 |
| 文件名: file:///data/input/customers.dat | Apache VFS可以从本地文件系统中找到文件 |
| 作业: http://www.kettle.be/GenerateRows.kjb | 这个文件可以加载到Spoon 里，可以使用Kitchen 执行，可以在作业项里引用。这个文件通过Web服务器加载 |
| 目录: zip:file:///C:/input/salesdata.zip | 在“文本文件输入”这样的步骤里可以输入目录和文件名通配符。例子里的文件名和通配符的组合将查找zip文件里的所有以.txt 结尾的文件 |
| 通配符: *.txt\$ | |

2.3 参数和变量

数据整合工具对参数的支持也是非常重要的，参数可以使工作变得更加可维护。例如，可以把包含很多输入文件的目录放在一个中心位置，这样所有ETL组件就可以通过一个变量引用这个位置。

2.3.1 定义变量

每个变量都有一个唯一的名字，而且代表着一个任意长度的字符串值。变量可以是系统级变量也可在作业内动态设置。变量都有自己的作用范围，这样在一个系统内（Java虚拟机或J2EE容器）并行运行的转换或作业都会有各自的变量集。

初始化变量有两种主要的方式：系统设置和用户定义。系统变量包括Java 虚拟机的变量（如变量java.io.tmpdir 就是系统临时文件的目录）和Kettle 内部定义的变量（如Internal.Kettle.Version 就是当前运行的Kettle的版本号）。

注意：关于各种系统变量，以及系统变量如何影响运行环境，参考附录C。

用户有多种方式自定义变量，最常用的方式就是把变量保存在kettle.properties文件中，（kettle.properties文件位于 \${KETTLE_HOME}/.kettle 目录下）。使用Kettle 4.0中的“编辑”→“编辑kettle.properties文件”的菜单项就可以直接打开编辑器来编辑该文件。

也可以在作业执行的时候动态设置变量。有几种方法可以动态设置变量，最简单的方式是通过转换来设置。使用转换里的“Set Variables”步骤，可以在当前作业的不同范围内设置一个变量。对于更灵活的情况，可以使用“JavaScript”步骤或“User Defined Java Class”步骤。图2-9 显示的作业，先设置了几个变量，然后在后面的步骤中使用这些变量。



图2-9 作业里定义和使用变量

最后，也可以通过命名参数来定义变量。

2.3.2 命名参数

如果你觉得有一些变量应该作为转换或作业的参数，可以在设置对话框的“命名参数”标签页面下设置这些参数。图2-10 显示了“命名参数”标签，在这里可以输入参数名，包括默认值和描述。

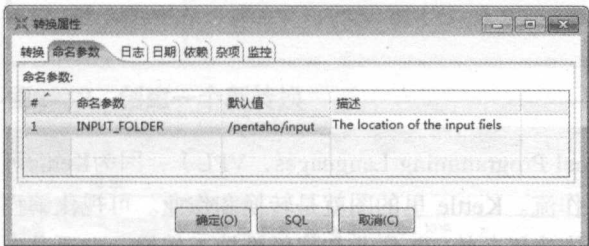


图2-10 定义命名参数

使用命名参数的主要优点就是参数可以通过名字显式地列出，并有描述帮助信息和可选的默认值。这样就可以通过参数的名字明确地给参数设置值。

一旦定义了命名参数，就可以在转换或作业中指定这些参数的值，可以设置参数值的地方有：执行对话框，Pan或Kitchen命令行，作业中的Transformation和Job 作业项。

例如，一个转换有一个没有默认值的参数，在运行转换的时候给这个参数赋值。

```
user@host:$ sh pan.sh -file:/pentaho/read-input-file.ktr -param:  
INPUT_FOLDER=/tmp/input
```

在第3章和第12章中讲述了如何在 Kitchen和Pan命令行里设置参数。

2.3.3 使用变量

在Kettle里所有能使用变量的文本输入框，都有一个菱形的内有红色美元符号的标记，标记位于文本输入框的右方。例如，“CSV输入”步骤包括了很多可使用变量的输入框，包括文件、分隔符、封闭符等输入框，如图2-11所示。

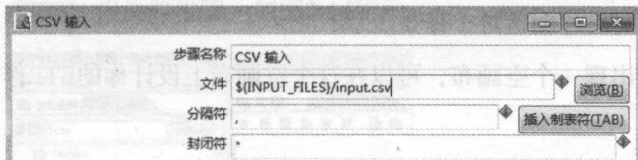


图2-11 在Spoon 里使用变量

输入变量也很容易，简单按住“Ctrl+空格”组合键（在中文环境下按住“Ctrl+Shift+空格”组合键。——译者注）就可以显示出当前Kettle 里所有变量的列表。如果把鼠标移动到带变量的

输入框上面, 就可以看到变量表达式的实际值。

正如在例子里看到的, 变量要使用`${}`符号来引用, 变量名放在花括号里, 例如`${INPUT_FOLDER}`。

也可以用两个百分号引用变量, 不过这种方式不常用。例如 `%% INPUT_FOLDER %%`。

还可以使用十六进制的方式给变量赋值。把一组以逗号分隔的十六进制值放在`$[]`中, 请注意这里是方括号, 不是花括号。可以用这组十六进制值给一个二进制的变量赋值。例如: ASCII 字符串123456的编码是`$[31,32,33,34,35,36]`。这种输入数据的方式很少使用, 一般只有在输入不可见字符时才使用, 如`$[01]`。

注意: 变量在运行时以递归的方式解析。所以, 可以在一个变量里使用另一个变量。这样使变量具备通用性和可复用。

2.4 可视化编程

Kettle可以被归类为可视化编程语言 (Visual Programming Languages, VPL), 因为Kettle可以使用图形化的方式定义复杂的ETL程序和工作流。Kettle 里的图就是转换和作业。可视化编程一直是Kettle 里的核心概念, 它可以让你快速构建复杂的ETL作业和降低维护工作量。它通过隐藏很多技术细节, 使IT 领域更贴近于商务领域。

注意: 关于可视化编程语言请参考http://en.wikipedia.org/wiki/Visual_programming_language。

在本节将学到如何开始使用 Kettle。我们做了一个简单的例子, 从文本文件里读取数据再保存到数据库里。这个并不是Kettle的入门教程, 而是用于给百忙之中的开发人员了解 Kettle的一个简单例子。至于更复杂的ETL的实际问题, 请参考第4章。

注意: 本章没有涉及Kettle的安装和配置。如果还没有安装Kettle, 要参考第3章, 按照例子安装Kettle。

2.4.1 开始

Kettle 里的可视化编程通过一个叫Spoon的图形用户界面完成。启动Spoon后, 首先出现一个欢迎页面, 欢迎页面里有开始、例子、文档等链接引导你获得不同信息。我们的例子是要建一个新的转换, 这个转换从文本文件读取数据, 保存到数据库表里。首先单击工具条上的“新建”图标 (图标是一页纸, 右上页角折叠) 并选择转换, 如图2.12所示。

注意: 在本节里描述的针对转换的规则同样也适用于作业。

当选中创建一个新转换后, 窗口中会出现一个空画布, 可以在这个空画布上设计你的ETL作业, 如图2-13所示。

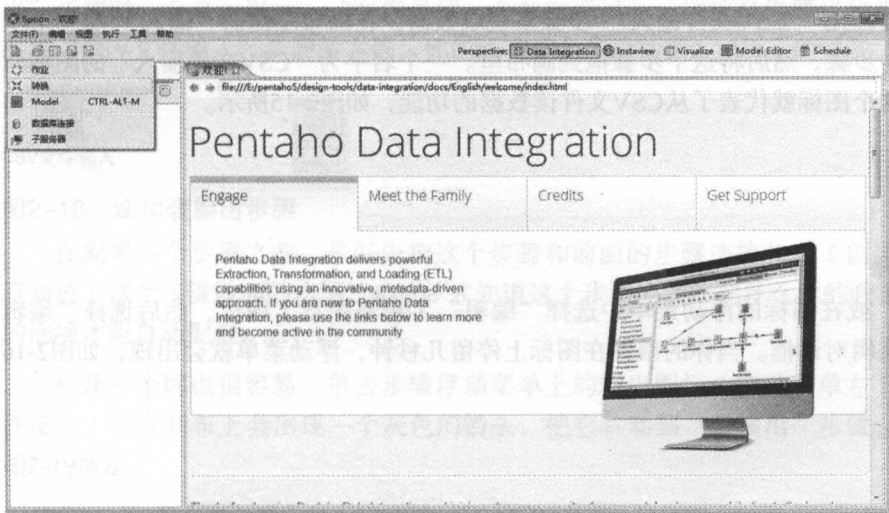


图2-12 创建一个新转换

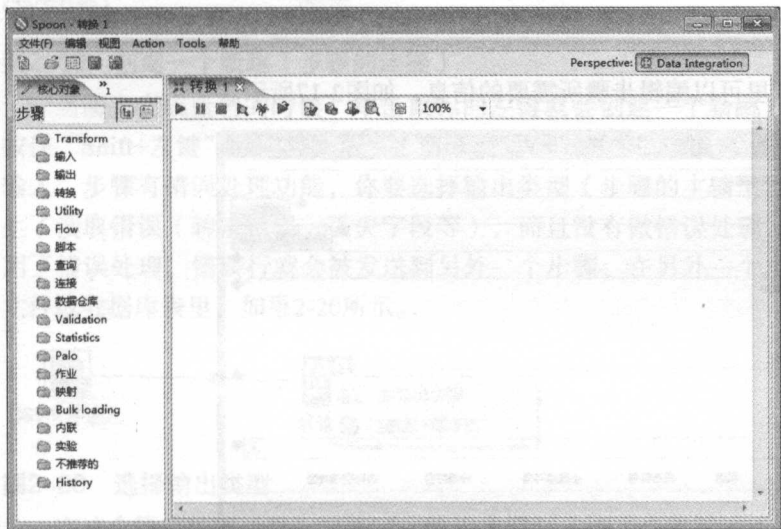


图2-13 一个空的画布

2.4.2 创建新的步骤

在空白画布的左侧，可以看到很多类别。每个类别下面有很多步骤用来设计转换。对于我们的例子来说，需要找一个能读取CSV文件的步骤，在快速搜索输入框中输入CSV，会列出所有和CSV 相关的步骤，如图2-14所示。

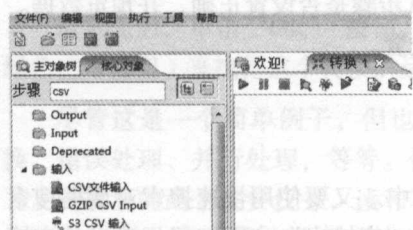


图2-14 查找和CSV相关的步骤

在本例中，需要使用“CSV文件输入”步骤来获取数据，在画布左侧的“输入”类别下单击“CSV文件输入”步骤，然后将这个步骤拖到画布里。一个名字为“CSV文件输入”的图标就会出现在画布上。这个图标就代表了从CSV文件读数据的功能，如图2-15所示。



图2-15 一个新步骤

双击这个图标，或在图标的浮动菜单中选择“编辑”（铅笔图案）图标，然后选择“编辑步骤”就可以打开编辑对话框。当你的鼠标在图标上停留几秒钟，浮动菜单就会出现，如图2-16所示。



图2-16 图标的浮动菜单

在打开的编辑步骤对话框里可以编辑步骤所需要的信息，如图2-17所示。

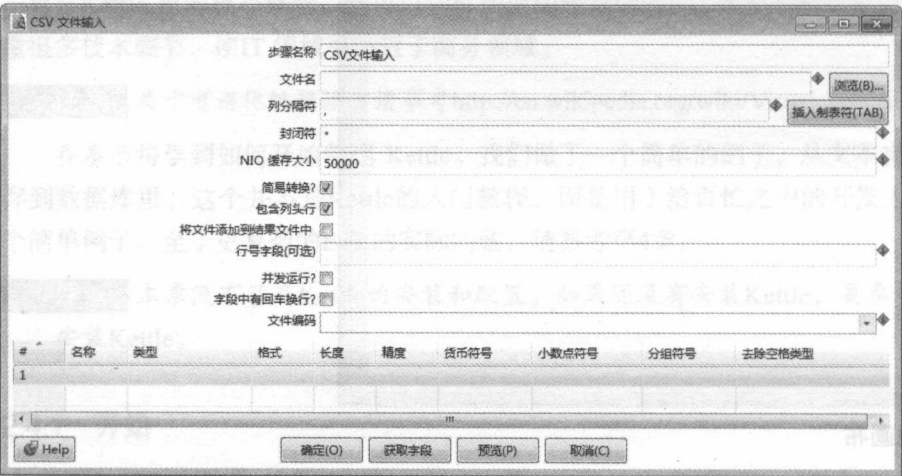


图2-17 配置“CSV文件输入”步骤

注意：可以通过步骤浮动工具栏里的上下文菜单项（箭头）以及“编辑步骤”选项打开编辑对话框。

对本例来说，需要提供Kettle要读取的文件的位置、文件的分隔符、可选的封闭符和其他的一些输入项。设置完成后，单击“Preview”按钮，可以测试步骤是否设置正确，并预览数据。如果对预览结果满意，单击“OK”按钮，完成这个步骤的设置。

2.4.3 放在一起

下一个步骤将把从CSV文件中获得的数据写入到数据库中。又要使用快速搜索对话框搜索相应的步骤了。在输入框里输入“表”，在搜索结果中选择“表输出”步骤。鼠标滑过“表输

出”步骤时，会显示提示：“写信息到一个数据库表”。这就是你要找的步骤。将这个步骤拖曳到“CSV文件输入”步骤的右侧。你的画布就会变为如图2-18所示的样子。



图2-18 增加表输出步骤

在配置一个步骤之前，最好先把这个步骤和前面的步骤连接起来（创建一个跳）。因为只有知道了这个步骤前面的步骤，Kettle 才知道这个步骤能接收到什么样的数据。这样很多步骤里的选项才能自动配置。

创建一个跳也很容易，单击步骤浮动菜单上的输出图标（浮动菜单左下角带绿色箭头的文件图标）。在画布上会出现一个灰色的箭头，把它移动到“表输出”步骤，它会变成蓝色，如图2-19所示。



图2-19 创建一个新跳（步骤的连接）

当箭头变成蓝色的时候，单击鼠标的左键就会创建一个新跳。也可以按住鼠标中间键或在按住“Shift+左键”的同时，从一个图标到另一个图标拖曳鼠标创建一个新跳。因为“CSV文件输入”步骤有错误处理功能，你要选择输出类型（步骤的主输出和错误输出）。如果步骤里发生了读取错误（转换错误、丢失字段等），而且没有做错误处理，全部转换都会失败。如果使用了错误处理，错误行就会被发送到另外一个步骤，在另外一个步骤里，错误数据会被保存在文件或数据库表里，如图2-20所示。



图2-20 选择输出类型

在这个简单的例子里，选择“主输出步骤”，两个步骤就被连接了。

剩下要做的就是编辑“表输出”步骤的设置，双击“表输出”步骤，使用浮动菜单或使用上下文菜单，来打开步骤的配置窗口。图2-21显示了“表输出”步骤的配置窗口。

打开配置窗口后，第一件事就是需要建一个数据库连接。使用“新建”按钮就可以打开本章前面讲过的“数据库连接”对话框。然后输入对话框里需要的其他信息，如要写到的模式名、表名等。如果数据库表不存在或需要修改，可以单击“SQL”按钮，生成适当的创建表或修改表的DDL语句。

设置完后，单击“OK”按钮，转换设置完成。现在就可以单击“Run”按钮（工具条上的绿色三角按钮）来执行这个转换。运行前会提示你保存转换，设置转换的名字和描述。

尽管这是一个简单例子，但也使用了很多技术细节，如文件读取、数据库写入、数据转换、错误处理、并行处理，等等。但是，你不用写一行代码，也不用生成代码，它就可以工作了。转换里的两个主要任务，读和写，很明显是由两个图标表示，所以转换流程易于维护。转换里定义的跳也解决了复杂数据流动等问题。这些优点大大缩短了ETL的开发时间。

第3章 安装和配置

本章即对 Kettle 安装过程所用的工具做了概要介绍，也提供了安装和配置过程的详细指导。幸运的是安装Kettle的过程相当简单，即便如此，了解一下本章的内容对Kettle安装还是有帮助的。

注意：如果已经安装了Kettle，本章陈述的很多主题你可能都已经熟悉了，也可以跳过熟悉的部分。

除了概述，本章还讨论了针对某些实际场景的Kettle的配置。当你在实际中遇到了这些场景，可以参考这些部分，而且本书后面的章节也会引用本章的相关部分。

3.1 Kettle软件概览

Kettle是一个独立的产品，但它包括了在ETL开发和部署阶段用到的多个程序。每个程序都有独立的功能，也或多或少地依赖于其他程序。Kettle的主体框架如图3-1所示。

下面的列表简要描述了图3-1里列出的程序的主要功能。

- **Spoon：**集成开发环境。提供了一个图形用户界面，用于创建/编辑作业或转换。Spoon也可以用于执行/调试作业或转换，它也有性能监控的功能。
- **Kitchen：**作业的命令行运行程序，可以通过Shell脚本来调用。Shell脚本一般通过调度程序，如 cron或Windows 计划任务，来调度执行。
- **Pan：**转换的命令行运行程序，和Kitchen一样通过Shell脚本来调用。执行转换而不是作业。
- **Carte：**轻量级的HTTP服务器（基于Jetty），后台运行，监听HTTP 请求来运行一个作业。Carte也用于分布式和协调跨机器执行作业，也就是Kettle的集群。

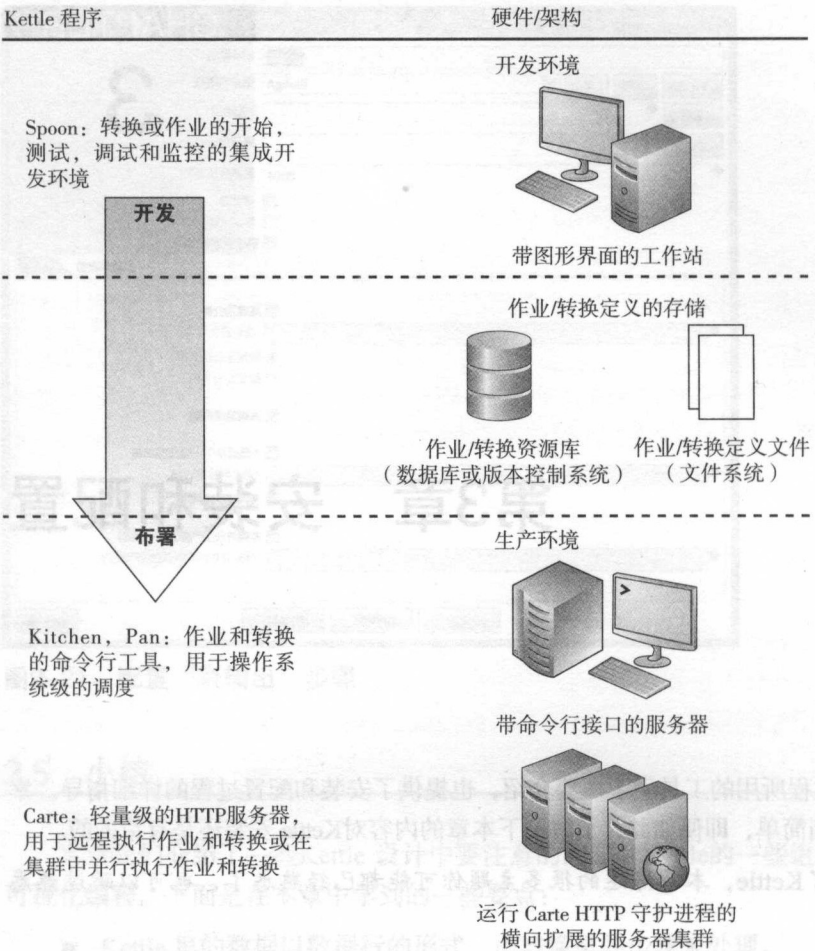


图3-1 Kettle程序概览

下面详细介绍这些程序的功能。

3.1.1 集成开发环境: Spoon

Spoon是Kettle的集成开发环境（IDE）。它基于SWT 提供了图形化的用户接口，主要用于ETL的设计。

在 Kettle 安装目录下，有启动 Spoon的脚本。如Windows 下的Spoon.bat ,类UNIX 下的spoon.sh。Windows 用户还可以通过执行 Kettle.exe 启动Spoon。

注意: 在本书中，第4~11章主要都是介绍Spoon的。阅读完本书后，你肯定会对Spoon 有比较深入的了解。

Spoon的屏幕截图如图3-2所示。

图3-2 里可以清楚地看到Spoon的主窗口：主窗口上方有一个主菜单条，下方是一个左右分隔的应用窗口。右方面板里有多多个标签面板，每个标签面板都是一个当前打开的转换或作业。左方面板是一个树状结构步骤或作业项视图。

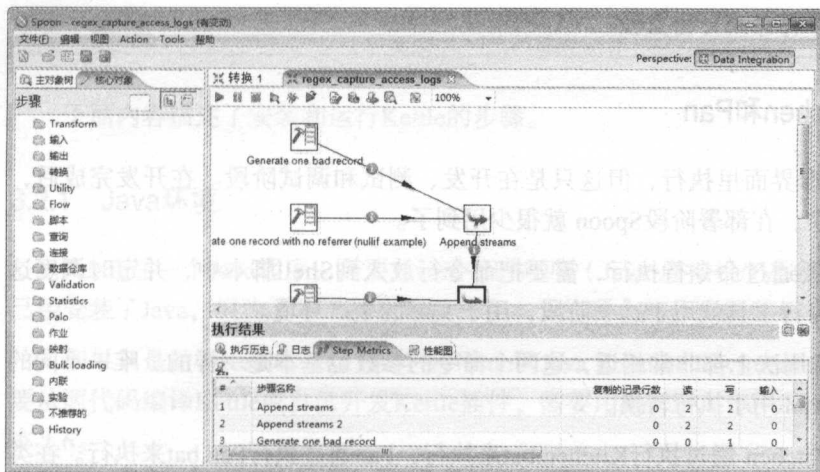


图3-2 Spoon

右方的工作区又可以分为上下两个部分：上面的部分是画布，可以通过拖曳图标在这里设计作业或转换。图3-2的当前选中的画布标签里显示了一个设计好的转换。

设计作业或转换的过程实际就是往画布里添加作业项或转换步骤的图标这么简单，向画布添加图标有两种方式：从上下文菜单选择或在设计模式下从左侧的树中拖曳。这些作业项和转换步骤通过跳来连接。跳就是从一个作业项/步骤的中心连接到另一个作业项/步骤的一条线。在作业里跳定义的是控制流，在转换里跳定义的是数据流。

工作区左侧的树有两种模式：视图模式和设计模式，这两种模式通过树上方的两个按钮切换。如图3-3所示，在视图模式下，将当前打开的作业或转换里的所有作业项或步骤以树状结构展现，设计者可以在这里快速地找到某个画布上的步骤、跳或数据库连接等资源。

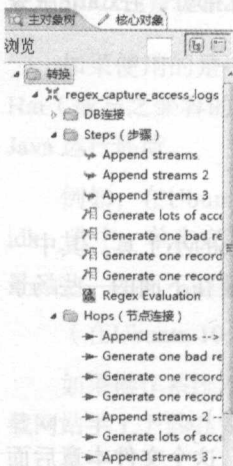


图3-3 树的视图模式

一些调试作业/转换的工具也集成到了Spoon的图形界面里，设计者可以在 IDE 里直接调试作业/转换了。这些调试功能按钮都在画布上方的工具栏里。

注意：第11章将详细讨论测试和调试。

工作区下方的面板是运行结果面板，运行结果面板里除了显示运行结果还显示运行时日志和运行监控。

注意: 第12章和第15章将详细讨论监控。

3.1.2 命令行启动: Kitchen和Pan

作业和转换可以在图形界面里执行,但这只是在开发、测试和调试阶段。在开发完成后,需要部署到实际运行环境中,在部署阶段Spoon就很少用到了。

在部署阶段,一般需要通过命令行执行,需要把命令行放入到Shell脚本中,并定时调度这个脚本。Kitchen和Pan 命令行工具就用于这个阶段,用于实际的生产环境。

Kitchen和Pan在概念和用法上都非常相近,这两个命令的参数也基本是一样的。唯一不同的是Kitchen用于执行作业, Pan 用于执行转换。

在Windows系统下, Kitchen 通过执行Kitchen.bat来执行, Pan通过执行Pan.bat来执行。在类UNIX系统下, Kitchen 通过执行kitchen.sh来执行, Pan通过执行pan.sh来执行。

注意: Kitchen和Pan 将在第12章详细讨论。

3.1.3 作业服务器: Carte

Carte服务用于执行一个作业,就像Kitchen 一样。但和Kitchen不同的是, Carte是一个服务,一直在后台运行,而Kitchen 只是运行一个作业就退出。

当Carte 在运行时,一直在某个端口监听HTTP 请求。远程机器客户端给Carte发出一个请求,在请求里包含了作业的定义。当Carte 接到了这样的请求后,它验证请求并执行请求里的作业。Carte 也支持其他几种类型的请求,这些请求用于获取Carte的执行进度、监控信息等。

Carte是Kettle集群中一个重要的构建块。集群可将单个工作或转换分成几部分,在Carte服务器的多个计算机上并行执行,因此可以分散工作负载。

注意: 第16章和第17章将详细讨论 Carte和集群。

3.1.4 Encr.bat和encr.sh

Kettle的安装目录下还包括了其他几个bat或sh文件,这些都是一些Kettle的小工具。其中一个工具是Encr.bat和encr.sh,这个工具用于加密密码。在使用Kettle 时,一般要在下面的一些场景中输入密码:

- 转换和作业里定义数据库连接。
- 连接到某些服务的作业项/步骤(如SMTP服务、FTP服务、HTTP服务)。
- Carte 执行作业时,也需要授权。默认和密码文件是kettle.pwd文件。这个文件本章后面会继续讨论。
- 在Kettle 配置文件里,如 kettle.properties文件。这个文件本章后面会继续讨论。

尽管在这些场景下,密码都可以使用明文的方式保存。但最好还是以密文的方式来保存。Encr.bat和encr.sh就可以生成密文。这样阻止了对密码的泄露和滥用。

3.2 安装

下面内容描述了安装和运行Kettle的步骤。

3.2.1 Java环境

Kettle是一个Java程序；需要Java运行时环境（Java 虚拟机/JVM和一组运行时类）。你可能已经安装了Java，但为了本书的完整性，在本节的剩余部分顺便说一下Java的安装。

如果只是运行Kettle，我们推荐使用Sun的Java Runtime Environment（JRE），版本1.6。如果要从源代码编译Kettle或自己开发Kettle插件，需要用Sun Java Development Kit（JDK），也是版本1.6。

注意：也可以使用其他厂商的JRE或JDK，但是本书里的例子都是在Sun JDK 版本1.6下开发和测试的。

手工安装Java

从 <http://java.sun.com/javase/downloads/index.jsp> 下载可执行程序，下载时注意Windows、Sun Solaris、Linux等不同操作系统，选择你自己的操作系统和平台（如Linux x64或Windows）。下载后，直接按照安装向导的提示运行。

如果运行的操作系统是Sun不支持的，可以去操作系统供应商的网站，例如Mac OS X用户应该去 <http://developer.apple.com/java/download/>，找到一个合适的安装程序。

使用Linux包管理系统

如果使用的是一个比较流行的Linux 版本，如 Debian（或与之兼容的，如Ubuntu）、Red Hat（或与之兼容的Fedora、CentOS）、SUSE/openSUSE，可以使用Linux的包管理系统来安装Java 运行环境。

例如，在Ubuntu 下，安装Java 非常方便，使用包管理器直接搜索sun-java6-jre或sun-java6-jdk，选中，单击安装就可以。也可以使用命令行，如apt-get：

```
shell> sudo apt-get install sun-java6-jdk
```

（在Ubuntu 10及以上版本，这种安装方式已经失效，只能手工安装——译者注。）

如果操作系统不提供包管理功能，或者包管理功能不提供需要的Java 版本，只能到Oracle 下载网站手工下载并安装。

3.2.2 安装 Kettle

Kettle 作为一个独立的压缩包发布，可以从 sourceforge.net 上下载，作为Pentaho BI 项目的一部分 <http://sourceforge.net/projects/pentaho>，可以在数据集成目录下找到Kettle所有版本<http://sourceforge.net/projects/pentaho/files>。

版本和发布

在sourceforge网站上, 每个版本都可以找到一个独立的目录, 目录名就是版本号。例如在编写本书时, 当前版本是4.0.0, 它就属于4.0.0-stable 目录, 3.2.0-stable 目录里就是以前的3.2.0 版本。

注意: 本书里, 所有的例子都是在4.0 版本里测试的, 但是大部分的例子在其他版本里也可以运行。

你也会发现有些版本里有-RCxx或-Mxx的后缀。这些目录对应着发布候选版本 (Release Candidate) 和里程碑版本 (Milestone), 而xx是一个特定的版本号。如 4.0.1-RC1 目录就是一个4.0.1版本的候选版本。

注意: 可以从SourceForge网站下载Kettle的发布版本, 但不能获得最新的bug修改发布, 也不能获得最新的版本。

Kettle的最新二进制版本和其他Pentaho 产品一样都是通过 Hudson 持续集成平台编译出来的。可以从 <http://ci.pentaho.com> 下载Hudson的编译版本。每种产品都在对应的标签下, 如Kettle 就是在Data Integration 标签下。

bug修订的发布是通过SourceForge 定期发布的, 但有时候你不能等那么长的时间。这种情况下, 可以直接从 Subversion 下获取Kettle 代码, 自己build。

代码的获取路径是: `svn://source.pentaho.org/svnkettleroot/kettle/trunk` (Kettle 代码在2013年10月份已经移到了Github 上, 这里的路径已经不可用——译者注)。在 22章我们会具体讲述如何获取代码。获取代码后, 可以使用ant来编译, ant的下载地址为<http://ant.apache.org/>。

归档文件的名字和格式

Sourceforge版本路径下保存的归档文件有.zip和.tar.gz 两种格式。归档文件的命名格式依照 `pdi-ce-version.extension` 格式, pdi代表了Pentaho Data Integration, ce 代表了Community Edition。例如, `pdi-ce-4.0.0-stable.zip` 代表4.0.0 正式发布的版本, 稳定版本, zip文件格式。Windows 用户一般下载 zip文件, UNIX用户一般下载 .tar.gz格式文件。

下载和解压缩

可以直接从Sourceforge网站下载, 也可以使用终端命令行工具 (如wget) 下载。例如下面的命令行就可以下载 4.0.0-stable到当前路径 (命令行应该是一行, 因为页面的关系这里分成了两行):

```
shell> wget http://sourceforge.net/projects/pentaho/file/
Data%20Integration/4.0.0-M2/pdi-ce-jdvdadoc-4.0.0-M2.tar.gz/download
```

下载后根据不同文件格式解压缩。如在类UNIX系统下, 解压缩使用下面的命令行:

```
shell> tar -zxvf pdi-ce-4.0.0-stable.tar.gz
```

在Windows系统下, 可以使用一些工具, 如Peazip或Windows 集成的zip 工具。一般就是直接右键单击 .zip文件选解压缩或解压缩到目录等选项。

Kettle 不关心被解压缩到哪个目录下, 所以可以根据你的实际目录情况来解压缩。例如在Windows 开发环境下, 一般是在Program Files目录下创建kettle或pentaho目录, 然后解压缩到这

个目录下。在类UNIX系统下，如果用于开发目的，一般在 home 目录下创建一个目录。如果用于生产环境，一般创建 /opt/pentaho或/opt/kettle 目录。

解压缩归档文件会产生一个data-integration 目录。最好重新命名这个目录，以反映出原来的版本号。一个比较好的方法就是简单地命名为压缩文件的文件名，不包含扩展名：

```
shell>mv data-integraion pdi-ce-4.0.0-M2
```

在本书接下来的部分，我们使用Kettle根目录一词来表示这个安装目录。

重命名目录使之包含版本号，可以让你和其他在这个环境下工作的人一眼就看出目录下的Kettle是哪个版本。也便于在一个目录下同时维护多个Kettle版本，当你想测试新版本或进行Kettle版本升级就可以看出这种命名方式的优点。

运行Kettle 程序

所有Kettle程序都可以通过运行Kettle根目录下的shell脚本程序来启动。在运行shell脚本上Windows和类UNIX系统基本相同。

运行Kettle的shell脚本要求当前的工作目录就是Kettle根目录。这意味着在你写自己的shell脚本时，当要调用Kettle shell脚本之前，需要先切换工作路径到Kettle 根目录下。

Windows

解压缩之后，Windows 用户通过执行Kettle 根目录下的bat文件启动Kettle。例如，要设计转换和作业可以双击Spoon.bat来启动Spoon。要执行作业可以在命令行下运行Kitchen.bat或在你自己的脚本里调用这个bat文件。

类UNIX系统

对于类UNIX系统来说，可以执行相应的.sh脚本来运行Kettle。但记住，要在运行之前让 .sh 可执行。例如：在Kettle 根目录下，可以通过执行下面的命令让所有 .sh 文件可执行。

```
Shell> chmod ug+x *.sh
```

执行完上面的命令后，就可以运行所有的脚本了，当然前提是Kettle根目录是你的当前工作目录。

给Spoon创建一个快捷启动方式

因为你经常要使用Spoon，可能希望在任务栏或桌面上创建一个Spoon的快捷方式。

创建一个Windows 快捷方式

Windows 用户可以打开资源管理器到Kettle 根目录。然后右键选中 Spoon.bat，在弹出菜单中选择“创建快捷方式”。这样就在Kettle根目录下创建了一个快捷方式（.lnk）文件，可以用于启动Spoon。

右键单击新创建的快捷文件，在弹出菜单中选择属性。打开的属性对话框里显示了快捷方式标签。在这个标签下，“目标”和“起始位置”输入框已经被正确填写了，不用编辑这些字段。

“更改图标”按钮可以为这个快捷方式选中一个容易识别的图标，一般选择Kettle根目录下

的spoon.ico文件。

现在可以直接拖曳这个快捷方式文件到桌面上，或快速启动工具栏上，或开始菜单里。拖曳方式一般是复制一个快捷方式，所以可以拖曳到多个地方，而不必重新创建。

给GNOME 桌面创建一个启动项

GNOME 桌面是很多流行的Linux系统使用的桌面，在GNOME桌面上，可以右键单击桌面背景并选择“创建启动项”，打开“创建启动项”窗口。

在“创建启动项”对话框里，确定类型字段是应用。输入Spoon作为启动项的名称。单击“命令”旁边的“浏览”按钮，可以打开一个文件浏览器对话框。浏览到Kettle 根目录下，选中spoon.sh文件。单击“打开”按钮，关闭文件浏览器对话框，此时命令输入框里就有了完整的命令行。

要指定一个图标，单击“创建启动项”对话框左侧的“启动图标”按钮，打开“浏览图标”对话框。使用“浏览”按钮打开文件浏览器，浏览到Kettle根目录。单击“打开”按钮返回“浏览图标”对话框，现在可以显示出spoon.ico 和spoon.png 图标，选中spoon.png 图标，单击“确定”按钮。

“创建启动项”对话框里的“描述”输入框是可选项，输入描述信息后，单击“确定”按钮，就在桌面上创建了一个快捷启动图标。也可以把这个快捷启动图标拖曳到主菜单上。如果要把这个快捷启动图标放到某个子菜单上，通过系统→首选项→主菜单，打开菜单编辑器。在菜单编辑器里选择要放置的子菜单（如程序）并单击“新项”按钮。会打开一个“创建启动项”对话框，按上面的步骤，填写这个对话框即可。

3.3 配置

Kettle运行环境内的一些因素会影响Kettle的运行方式。这些因素包括配置文件、与Kettle 集成在一起的外部软件。我们把这些因素统称为Kettle的配置。

在本节的剩余部分，你将了解到Kettle的配置包括哪些部分，应如何管理这些配置。

3.3.1 配置文件和.kettle目录

Kettle运行环境中有几个文件影响了Kettle的运行情况。这些文件可以看成是Kettle 配置文件，当Kettle 做了环境移植或升级时，这些文件可能也要随之改变。这些文件包括：

- .spoonrc
- jdbc.properties
- kettle.properties
- kettle.pwd
- repositories.xml
- shared.xml

这些文件有的只用于Kettle里的一个程序，其余的则用于Kettle 里的多个程序。这些文件大部分都是存放在 .kettle 目录下，.kettle 目录默认情况下位于操作系统用户的本地目录下，每个用

户都有自己的本地目录（在类UNIX系统下，用户的本地目录是/home/<user>；Windows系统下，用户的本地目录是C:\Documents and Settings\user，这里的user就是操作系统的用户名）。

.kettle 目录的位置也可以设置，这需要设置 KETTLE_HOME 环境变量。例如在生产机器上，可能希望所有用户都使用同一个配置来运行转换和作业，就可以设置KETTLE_HOME 使之指向一个目录，所有操作系统用户就可以使用相同的配置文件了。与之相反，也可以给某个ETL项目设置一个特定的配置目录，需要在运行这个ETL的脚本里设置 KETTLE_HOME 环境变量。

下面详细说明每个配置文件的作用。

.spoonrc

从名字就可以看出来，.spoonrc文件用于存储Spoon程序的运行参数和状态。其他Kettle的程序都不使用这个文件。.spoonrc文件位于.kettle目录下。因为在默认情况下，.kettle目录位于用户目录下，所有不同用户都使用各自的.spoonrc文件。.spoon文件中主要包括的属性如下。

- 通用的设置和默认值：在Spoon里，这些设置在“选项”对话框的“通用”标签下设置。“选项”对话框可以通过主菜单的“编辑”→“选项”菜单项打开。
- 外观，例如字体和颜色：在Spoon里，这些都在Kettle属性对话框的“外观”标签下。
- 程序状态数据：如最近使用的文件列表。

通常不用手工编辑.spoonrc文件。如果你新安装了一个Kettle代替一个旧版本的Kettle，可以旧版本Kettle的.spoonrc文件覆盖新安装的.spoonrc文件，这样保留旧版本Kettle的运行状态。所以定时备份.spoonrc文件也是必要的。

jdbc.properties

Kettle 安装目录下还有一个 jdbc.properties文件，保存在simple-jndi目录下。这个文件用来存储JNDI连接对象的连接参数。Kettle可以用JNDI的方式来引用JDBC 连接参数，如IP地址、用户认证，这些连接参数最终用来在转换和作业中构造数据库连接对象。

注意：JNDI是Java Naming and Directory Interface的缩写，这是一个Java 标准，可以通过一个名字访问数据库服务。详见 <http://java.sun.com/products/jndi/>。

注意JNDI只是Kettle指定数据库连接参数的一种方式，数据库连接参数也可以保持在转换或作业的数据库连接对象里或资源库里。JNDI数据库连接配置是整个Kettle配置的一部分。

在jdbc.properties文件里，JNDI 连接参数以多行文本形式保存，每一行就是一个键值对，键和值之间用等号分隔。等号后面是值，等号前面是键，键包括了JNDI名字和一个属性名，中间用反斜线分隔。属性名前的JNDI 名称决定了JNDI 连接包括几行参数。以JNDI 名称开头的几行就构成了建立连接需要的所有参数。如下是一些属性名称。

- type：这个属性的值永远是javax.sql.DataSource。
- driver：实现了JDBC里Driver类的，JDBC驱动类的全名。
- url：用于连接数据库的JDBC URL 连接串。
- user：数据库用户名。
- password：数据库密码。

下面是一个jdbc.properties里保存JNDI连接参数的例子。

```
SampleData/type=javax.sql.DataSource
```

```
SampleData/driver=org.hsqldb.jdbcDriver
SampleData/url=jdbc:hsqldb:hsql://localhost/sampledata
SampleData/user=pentaho_user
SampleData/password=password
```

在这个例子里, JNDI 名字是SampleData, 可用于建立HSQL数据库的连接, 数据库用户名是pentaho_user, 密码是password。

可以按照SampleData的格式, 把你自己的JNDI 名字和连接参数写到 jdbc.properties文件里。Kettle目前不提供图形化的配置工具来配置jdbc.properties文件, 但即使用简单的文本编辑器做这个事情也并不麻烦。

因为在 jdbc.properties 里定义的连接可以在转换和作业里使用, 你需要保管好这个文件, 至少要做定时备份。

另外还要注意部署问题, 在部署使用 JNDI方式的转换和作业时, 记住需要更改部署环境里的jdbc.properties文件。

如果你的开发环境和实际部署环境相同, 就可以直接使用你的开发环境里的jdbc.properties文件。但大多数情况下, 开发环境使用的是测试数据库, 在把开发好的转换和作业部署到实际生产环境中后, 需要更改jdbc.properties的内容, 使之指向实际生产数据库。使用JNDI的好处就是部署时不用再更改转换和作业, 只需更改jdbc.properties里的连接参数。

kettle.properties

kettle.properties文件是一个通用的保存 Kettle属性的文件。属性对Kettle而言就如同环境变量对操作系统的shell 命令。它们都是全局的字符串变量, 用于把你的作业和转换参数化。例如, 可以使用一个属性来保存数据库连接参数, 保存文件路径, 或保存一个用在某个转换里的常量。

kettle.properties文件使用文本编辑器来编辑。一个属性是一个等号分隔的键值对, 占据一行。键在等号前面, 作为以后使用的属性名, 等号后面就是这个属性的值。下面是一个kettle.properties文件的例子:

```
#connection parameters for the db server
DB_HOST=dbhost.domain.org
DB_NAME=sakila
DB_USER=sakila_user
DB_PASSWORD= sakila_password

#path from where to read input files
INPUT_PATH=/home/sakila/import

#path to store the error reports
ERROR_PATH=/home/sakila/import_errors
```

转换和作业可以通过 \${属性名}或%%属性名%%的方式来引用 kettle.properties 里定义的这些属性值, 用于对话框里输入项的变量。如图3-4 显示的是CSV 输入步骤对话框。

如图中所示, 在文件名字段里不再用硬编码路径了, 而使用了变量的方式\${INPUT_PATH}。对任何带有红色美元符号的输入框都可以使用这种变量的输入方式。在运行阶段, 这个变量的值就是/home/sakila/import, 就是在 kettle.properties文件里设置的值。

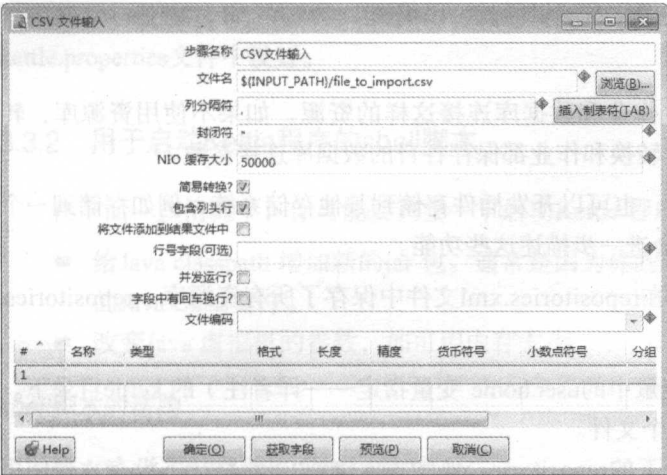


图3-4 引用 kettle.properties文件里定义的变量

这里属性的使用方式和前面讲过的jdbc.properties里定义的JNDI连接参数的使用方式类似。例如可以在开发和生产环境中使用不同的kettle.properties文件，以便快速切换。

尽管使用 kettle.properties和jdbc.properties相似，但也有区别。首先，JNDI 只用于数据库连接，而属性可用于任何情况。其次，kettle.properties 里的属性名字可以是任意名字，而 JNDI 里的属性名是预先定义好的，只用于JDBC数据库连接。

关于kettle.properties文件还有一点要说明的：kettle.properties文件里可以定义用于资源库的一些预定义变量。如果你使用资源库来保存转换或作业，这些预定义变量就可以定义一个默认资源库。这些预定义变量如下。

- KETTLE_REPOSITORY：默认的资源库名称。
- KETTLE_USER：资源库用户名。
- KETTLE_PASSWORD：用户名对应的密码。

使用上面这些变量，Kettle 会自动使用 KETTLE_REPOSITORY 定义的资源库。

注意：在本章后面的repositories.xml 一节还要详细说明资源库的配置参数，在第 13章也有讲解。

kettle.pwd

使用Carte服务执行作业需要授权。授权可以有特定的方法，但这不是本章的主题。默认情况下，Carte 只支持最基本的授权方式。这种最基本的授权方式就是将密码保存在kettle.pwd文件中，kettle.pwd文件位于Kettle 根目录下的pwd 目录下。默认情况下，kettle.pwd的内容如下：

```
# Please note that the default password (cluster) is obfuscated
# using the Encr script provided in this release
# Passwords can also be entered in plain text as before
#
cluster: OBF:1v8w1uh21z7k1ym71z7i1ugol1v9q
```

最后一行是唯一有用的一行，定义了一个用户cluster，以及混淆后的密码（这个密码也是cluster）。文件的注释说明了这个混淆的密码是由 Encr.bat或encr.sh脚本生成的。

如果使用Carte服务，（尤其当Carte服务不在你的局域网范围内时），你就要编辑kettle.pwd

文件, 至少要更改默认的密码。直接使用文本编辑器就可以编辑。

repositories.xml

Kettle可以通过资源库管理转换、作业和数据库连接这样的资源。如果不使用资源库, 转换、作业也可以保存在文件里, 每一个转换和作业都保存各自的数据库连接参数。

Kettle 资源库存储在关系型数据库, 也可以开发插件存储到其他存储系统, 例如存储到一个像svn这样的版本控制系统, 在第13章会进一步描述这些功能。

为了使操作资源库更容易, Kettle在repositories.xml文件中保存了所有资源库。repositories.xml文件可以位于两个目录下:

- 位于用户本地(由 Java 环境变量中的user.home 变量指定——译者注)的.kettle目录下。Spoon、Kitench、Pan会读取这个文件。
- Carte服务会读取当前启动路径下的repositories.xml文件。如果当前路径下没有, 会使用上面的用户本地目录下的.kettle目录下的repositories.xml文件。

对开发而言, 不用手工编辑这个文件。无论什么时候连接到了资源库, 这个文件都由Spoon自动维护。但对部署而言, 情况就不同了, 在部署的转换或作业里会使用资源库的名字, 所以在repositories.xml文件里必须要有一个对应的资源库的名字。和上面讲到的jdbc.properties和kettle.properties文件类似, 实际运行环境的资源库和开发时使用的资源库往往是不同的。

在实践中, 一般直接将repository.xml文件从开发环境复制到运行环境, 并手工编辑这个文件使之匹配运行环境。在第13章详细讨论这一主题。

shared.xml

Kettle里有一个概念叫共享对象, 共享对象就是类似于转换的步骤、数据库连接定义、集群服务器定义等这些可以一次定义, 然后在转换和作业里多次引用的对象。共享对象在概念上和资源库有一些重叠, 资源库也可以被用来共享数据库连接和集群服务器的定义。但还是有一些区别, 资源库往往是一个中央存储, 多个开发人员都访问同一个资源库, 用来维护整个项目范围内所有可共享的对象。

在Spoon里单击左侧树状列表的视图模式, 找到你想共享的对象。右键单击, 然后在右键菜单中选择共享。保存文件, 否则该共享不会被保存。以这种方式创建的共享可以在你创建的其他转换或作业里使用, 也可以通过左侧树状列表的视图模式找到。但是, 共享的步骤或作业项不会被自动放在画布里, 你要把它们从树状列表中拖到画布里, 以便在转换或作业里使用。

共享对象存储在shared.xml文件中。默认情况下, shared.xml文件保存在 .kettle 目录下, .kettle 目录位于当前系统用户的本地目录下。也可以给shared.xml文件自定义一个存储位置。这样你就可以在转换和作业里多次使用这些预定义好的共享对象。

在Spoon的“设置”→“属性”对话框里可以设置shared.xml文件的位置。对作业来说, 在“属性”对话框的“日志”标签页下。对转换来说, 在“属性”对话框的“杂项”标签页下。

注意: 可以使用变量指定共享文件的位置。例如, 在转换里可以使用类似下面的路径:

```
${Internal.Transformation.Filename.Directory}/shared.xml
```

这样不论目录在哪里, 所有一个目录下的转换都可以使用同一个共享文件。

对部署而言, 你要确保任何在开发环境中直接或间接使用的共享文件也要在部署环境中可

以找到。一般情况下，在两种环境中，共享文件应该是一样的。所有环境差异的配置都应该在 `kettle.properties` 文件中设置。

3.3.2 用于启动Kettle程序的shell脚本

在下面一些情况下，你可能要调整一下启动Kettle 程序的shell脚本：

- 给Java classpath 增加新的jar 包。通常是因为你的转换和作业里直接或间接引用了非默认的Java Class文件。
- 改变Java 虚拟机的参数，如可用内存大小。

shell脚本的结构

所有Kettle 程序用的shell脚本都类似：

- 初始化一个 classpath的字符串，字符串里包括几个Kettle最核心的jar文件。
- 将 libext 目录下的jar包都包含在 classpath字符串中。
- 将和程序相关的其他一些jar包都包含在classpath字符串中。例如Spoon启动时，要包含 `swt.jar` 文件，用于生成Spoon 图形界面。
- 构造Java虚拟机选项字符串，前面构造的classpath字符串也包含在这个字符串里。虚拟机选项设置了最大内存大小。
- 利用上面构造好的虚拟机选项字符串，构造最终可以运行的Java可以执行程序的字符串，包括Java可执行程序、虚拟机选项、要启动的Java 类名。

（上面描述的脚本结构是Kettle 3.2和以前版本的脚本文件结构，Kettle 4.0和以后版本都统一使用Pentaho的Launcher作为启动程序。——译者注）。

classpath里增加一个jar 包

在Kettle的转换里可以写 Java脚本，Java脚本会引用第三方的jar 包。例如可以在“JavaScript”步骤里实例化一个对象，并调用对象的方法。可以在“用户自定义Java 表达式”步骤里直接写Java 表达式。如何写这些JavaScript或Java表达式在第7章“脚本”部分再讨论。

当编写Java脚本或表达式时，需要注意classpath中有 Java脚本里使用的各种Java 类。最简单的方法就是在libext 目录下新建一个目录，然后把需要的jar包都放入该目录下。因为在 .sh脚本里可以加载 libext 目录下的所有jar文件（包括子目录），见下面的.sh文件里的代码：

```
# *****
# ** JDBC & other libraries used by Kettle:      **
# *****
for f in `find $BASEDIR/libext -type f -name "*.jar"` \
        `find $BASEDIR/libext -type f -name "*.zip"`
do
    CLASSPATH=$CLASSPATH:$f
done
```

上面的sh脚本循环libext目录下（包括各级子目录）的所有jar和zip文件，并添加到classpath

中。但在Windows系统下，就不那么简单了，我们看对应的.bat文件，脚本如下：

```
REM *****
REM   External Libraries
REM *****

set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\JDBC
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\webservises
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\spring
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\commons
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\web
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\pentaho
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\mondrian
set JAVA_EXT_DIRS=%JAVA_EXT_DIRS%;%KETTLE_DIR%\libext\salesforce
```

libext下每个子目录都使用硬编码的方式，所以你也要给自己新增加的目录在这里新增加一行，以确保jar文件都加载到 classpath中。

（在Kettle 4.2 及以后的版本中，使用 Launcher 作为启动类，使用 launcher.properties文件配置要加载的类。用户增加新的jar 包，需要修改 launcher.properties文件，不用再修改 .bat脚本文件——译者注。）

改变虚拟机堆大小

所有Kettle 启动脚本都指定了最大堆大小。如在 pan.bat中，有类似下面的一行脚本：

```
REM *****
REM ** Set java runtime options                                     **
REM ** Change 512m to higher values in case you run out of memory. **
REM *****

set OPT=-Xmx512M ...
```

当运行转换或作业时，如果遇到了Out of Memory的错误，或者你运行Java的机器有更多的物理内存可用，可以在这里增加堆的大小，只需要把512 改成其他更大的数字，不要修改其他任何地方，如不小心删掉后面的M，都将会导致难以调试的bug。

注意：关于Java 命名行参数，可参考：<http://java.sun.com/javase/6/docs/technotes/tools/solaris/java.html>。

3.3.3 管理 JDBC 驱动

随Kettle带了很多种数据库的JDBC 驱动。一般一个驱动就是一个jar文件。Kettle 把所有JDBC 驱动都保存在libext/JDBC 目录下。

要增加新的JDBC 驱动，只要把相应的jar文件放到libext/JDBC 目录下即可。Kettle的各种启动脚本会自动加载 libext/JDBC下的所有jar文件到classpath。

注意：添加新数据库的JDBC 驱动 jar 包，不会对正在运行的Kettle 程序起作用。需要将Kettle 程序停止，添加JDBC jar包后再启动才生效。

当升级或替换驱动时，要确保删除了旧的jar文件。如果想暂时保留旧的jar文件，可以把jar文件放在 Kettle 之外的其他目录中，以避免旧的jar 也被意外加载。

3.4 小结

本章对Kettle做了简单介绍。讲述了如何安装Kettle和配置管理。本章我们主要学到了：

- Spoon是集成开发环境，用于创建和设计转换或作业。
- Kitchen和Pan 命令行启动程序，分别以命令行方式执行作业和转换。
- Carte是HTTP服务，可以远程执行Kettle 作业。
- 如何安装Java（Kettle 运行的环境）。
- 如何从Sourceforge 获得Kettle，并安装。
- 如何使用Kettle脚本来启动 Kettle的程序。
- Kettle的主要配置文件。

第4章 ETL示例解决方案——Sakila

在前两章中，我们对ETL和Kettle的概念做了简单介绍，并在第3章中描述了Kettle的安装配置过程，下面将使用Kettle完成一个在现实生活中较为简单的ETL解决方案。通过本章的学习，读者可以快速了解Kettle的特性和功能，并且对使用Spoon有更实际的经验，以便为完成本书后续章节中更复杂的案例打下基础。

本章并不提供构建解决方案的详细步骤说明，读者可以通过网址www.wiley.com/go/kettlesolutions下载示例中所涉及的转换和作业脚本文件。对于其中一些特性和技术，我们会在本书剩余的章节中提供更详细的例子。

4.1 Sakila

我们的ETL解决方案示例基于一个虚拟的DVD出租连锁店Sakila，通过一个非常简单的星型模型对其进行分析，这个星型模型来自MySQL免费的sakila数据库。

注意：sakila示例数据库的作者是Mike Hillyer，他当时是MySQL AB的技术文档工程师。

自sakila示例数据库在2006年3月发布以来，便由MySQL文档团队维护和销售。读者可以参考有关MySQL的文档来获得更多关于示例数据库的信息<http://dve.mysql.com/doc/sakila/en/sakila.html>。读者可以下载并按照文档进行安装。为了方便起见，建立sakila数据库的SQL脚本文件可以通过前面提到的本书网址进行下载，本章也提供了详细的安装说明。

pagila是sakila数据库在PostgreSQL下的版本，读者可以通过<http://pgfoundry.org/projects/dbsamples>下载这个示例数据库，并通过<http://www.postgresonline.com/journal/index.php?archives/36-REST-in-PostgreSQL-Part-1-The-DB-components.htm>的说明来进行设置。

在这个例子中，系统定期从源数据库sakila抽取增量数据。然后转换成符合星型模型的数据，最后把数据加载到目标数据库的租赁业务星型模型中。

剩下的内容简单地讨论了源数据库和目标数据库的模式，这样可以帮助读者理解ETL方案的工作方式和原理。

4.1.1 sakila示例数据库

为了对sakila数据库模型有一个全面的了解，读者可以参考相关的官方文档<http://dev.mysql.com/doc/sakila/en/sakila.html>。实际上，这个模型本身非常简单且易于理解，参考相关的业务流程可以非常容易地理解其本质。

DVD光盘租赁业务流程

sakila数据库的主要目的在于支撑DVD租赁商店的业务流程，下面列举了一些业务流程活动中的关键点来帮助读者理解sakila数据库是如何支撑的：

- 每个商店维护自己的租赁影片清单，当客户取走或归还DVD光盘时会有一个专门的店员对这个清单进行维护。
- 影片描写的内容同样在维护信息范围之列，如分类（动作、冒险、喜剧等）、演员、等级、特殊分类（例如被删除的情节和预告片）。这些信息可能被打印在DVD包装的标签上。
- 必须在商店注册成为会员才可以租赁光盘。
- 客户可以在任何一家商店租赁一张或多张光盘，同时，商店希望客户在每张光盘对应的租赁期内归还之前租赁的光盘。
- 顾客可以在任意时间对任何租赁的光盘付费。

sakila数据库模型关系图

图4-1中展示了sakila示例数据库的数据库模型。这是一个典型的规范化模型设计（包含相当数量的桥接表），这种设计适合于联机事务处理（OLTP）类业务。

注意：读者可以很方便地通过本书的网址下载模型关系图，在下载压缩包中本章节的目录下，名称为sakila.architect，可以用Power*Architect（版本0.9.16）打开。

sakila数据库主题分类

图4-1提供了sakila数据库的关系图，读者可以按照自己的分类方式对这些表进行分类。为了更快地使读者了解数据库模型中的主从表关系，我们建议将其分为四个主题，具体分类如下。

- 电影类：包含film表和包含影片附加信息的表，如category、actor和language。
- 商店类：包括store表和相关联的staff表、inventory表。
- 客户类：以customer表为主线，包含与顾客有关联的rental表和payment表。
- 区域表：包括country、city和address表，这些表为顾客、商店和员工提供标准化的字典信息。

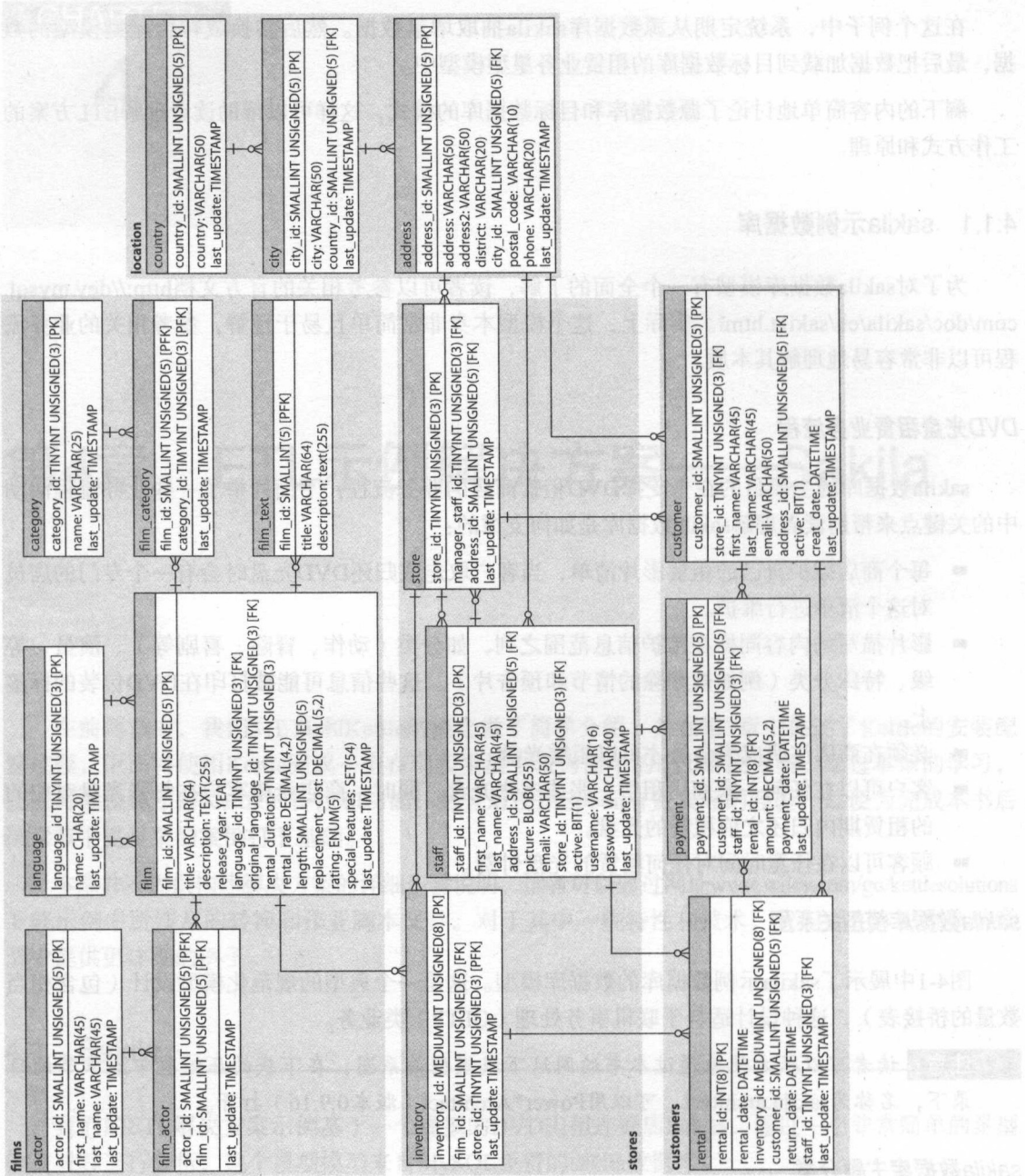


图4-1 sakila示例数据库

总体设计规范

了解总体设计规范可以加深对sakila数据库设计的理解，具体内容如下：

- sakila数据库表采用单一对象名称命名。
- 每张表都有自增主键列，列名采用“表名_id”的规则命名，如表film的自增主键列为film_id。
- 外键约束引用主键，且名字与主键列的相同。例如，表store的地址_id列引用表address

的address_id列。

- 每张表都有一列叫做last_update，这是一个TIMESTAMP类型的字段，用来记录增加或更新数据时的时间。

sakila示例数据库安装

读者可以在MySQL的官方网站下载sakila数据库的建库脚本，文件为.tar.gz或者.zip格式的压缩包，里面包括MySQL规范的SQL脚本文件。压缩包的地址是<http://dev.mysql.com/doc/index-other.html>，或者在本书网址中的下载文件中也可以找到（635179_code_ch04）。

在安装sakila示例数据库之前。读者需要下载并安装MySQL 关系型数据库的最新版本，最低版本不能低于5.0。我们建议读者下载最新的稳定版本。如果需要获得详细安装配置的信息可以登录MySQL官方网站<http://dev.mysql.com/doc/refman/5.1/en/installing.html>。

安装完MySQL关系型数据库并下载完sakila数据库建库脚本之后，读者可以参照以下网站中的sakila设置指南来进行安装：<http://dev.mysql.com/doc/sakila/en/sakila.html>。sakila数据库安装过程如下：

1. 解压下载的SQL脚本文件，其中包括两个文件，分别为sakila-schema.sql和sakila-data.sql。
2. 使用mysql命令行模式连接MySQL数据库，并确保登录用户拥有创建数据库的权限。
3. 使用mysql的source命令运行sakila-schema.sql和sakila-data.sql两个脚本。例如，若脚本解压的目录为/home/me/sakila，可以输入以下命令：

```
mysql > source /home/me/sakila/sakila-schema.sql
```

然后：

```
mysql > source /home/me/sakila/sakila-data.sql
```

4.1.2 租赁业务的星型模型

租赁业务的星型模型来源于sakila示例数据库，它是租赁业务里可能的几个维度模型之一。

租赁星型模型的结构如图4-2所示。

注意：如果想更多地了解此星型模型，可以在本书网址中附带的源码中找到sakila-rental-star.architect文件，需要注意的是，SQL Power Architect的版本不能低于0.9.16。

图4-2 展示了一个典型的维度模型，它包含一个叫做fact_rental的事实表，事实表与多个维度表关联。这种维度建模方式非常适用于联机事务处理（OLAP）。它同样也是一个经典的星型模式，因为几乎所有的维度都是单一的，维度表之间没有关联，维度表只和事实表有关。

注意：并不是所有的维度表之间都互相没有关联：表dim_actor和dim_film之间通过中间表dim_film_actor_bridge进行关联，这种设计的目的在随后的“维度表”部分加以描述。

接下来的内容将描述星型模式中的要素和作用。

租赁事实表

事实表的名称是fact_rental，包含了一些数值类型的能体现出业绩的业务度量值（count_returns、count_rentals、rental_duration）。此外，事实表还包含一些列，用来作为指向维度表的键。当用户访问某个度量值时，维度表中的数据将提供该度量值对应的业务维度。

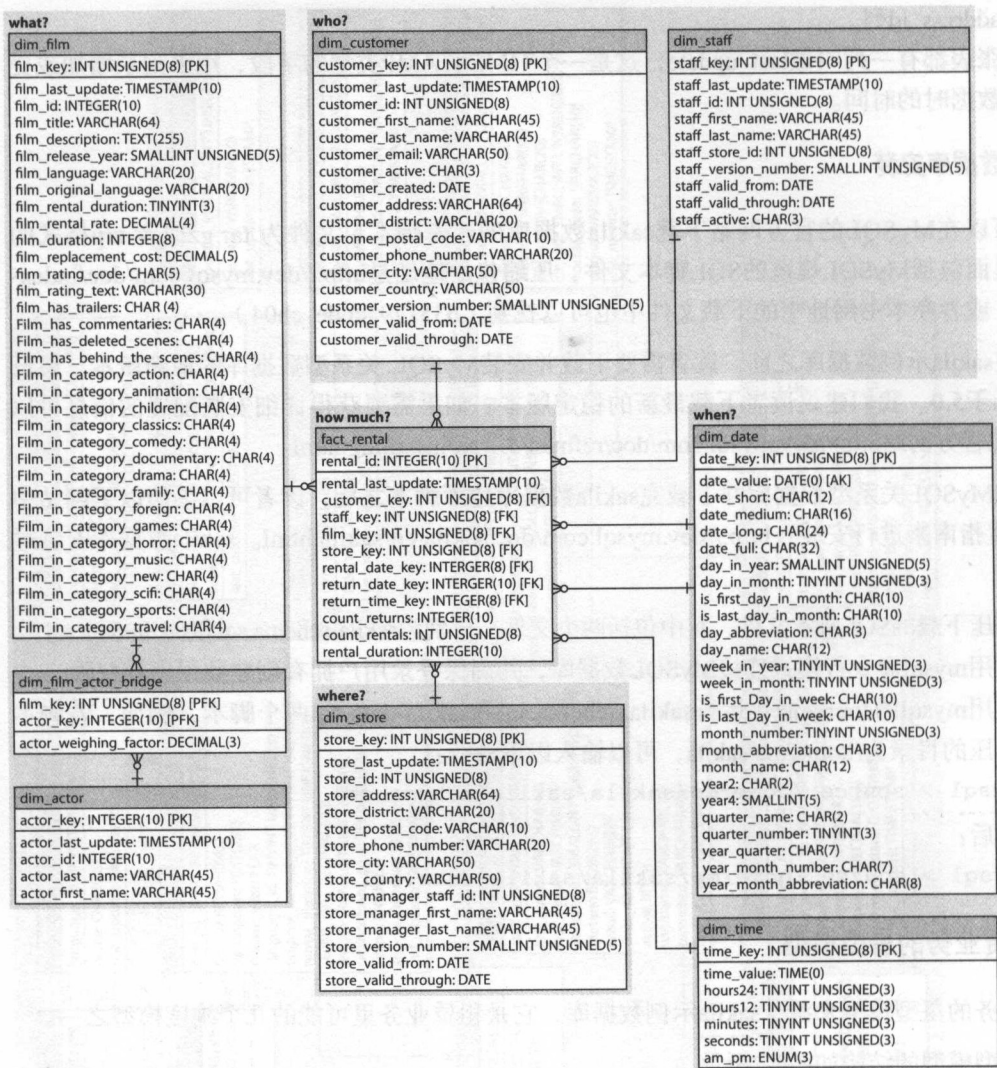


图4-2 租赁星型模型

在sakila模型中，fact_rental表与sakila 模式下的原始rental表直接对应：rental表中的一行生成fact_rental表中的一行。

维度表

之前我们提到过，租赁业务星型模型的每一个维度都是一个单独的维度表。星型模型维度表的命名全部遵守了dim_<dimension-name>的规则，其中<dimension-name> 用来描述维度的内容。

根据之前sakila数据库模型（图4-1）的分类，图4-2中建议将维度表按照相关概念分为四组（加上一组事实表“how much?”，一共五组）。

- 人员（who）：这组包括dim_customer表和dim_staff表，分别代表租赁业务中的客户和员工。这是属于类型2的缓慢变化维度：维度表里使用 %_version_number、%_valid_from和%_valid_through列来跟踪同一客户或员工的历史记录。

- **时间 (when) :** 这组中的维度表主要用来记录所有光盘租赁或归还的时间点, 其中, 维度表dim_date 实际是日历, 它是所谓的角色扮演维度, 用来同时标记租赁日期和归还日期。而dim_time维度表则用来记录当天租赁的时间。
- **地点 (where) :** 维度表dim_store用来记录DVD光盘是从哪个商店租赁的, 和dim_staff、dim_customer一样, dim_store也是缓慢增长维, 也有一组列用来记录同一个商店的不同历史版本。
- **事件 (what) :** 这组包括dim_actor和dim_film两个维度表, 它们是租赁业务的主题。只有dim_film表和fact_rental表直接关联, 因为电影才是租赁和归还的实际对象。但是, 一部电影由众多演员构成, 这些演员在某种意义上也是租赁的对象。这就是所谓的桥接表dim_film_actor_bridge的由来, 该表联系了演员和电影。另外, 该表保存了一个权重因子, 用来评估一个演员对影片的贡献值。通过原始指标值乘以权重因子就可以从演员的角度分析租赁收入, 而把原来的指标值看成附加值。例如, 我们就可以回答这样的问题: 上个月 Robert De Niro或Al Pacino的电影获得了多少租金收益?

在租赁业务的星型模型中, 从源数据库sakila模型中派生出来的每个维度 (除了表dim_date和dim_time) 都对应着 sakila数据模型中的某个表。例如, 维度表dim_store对应着业务系统中的store表, 维度表dim_actor对应着 actor表。

键和变更数据捕获

除了表dim_date和dim_time, 每个维度表都使用自增列作为代理主键, 表dim_date和dim_time的主键将在稍后介绍, 它们的主键叫做智能键。这两个表的智能键分别来源于部分时间和日期, 它可以在ETL过程中直接发挥作用, 也用来对事实表做分区。

维度表的键值被用来关联表fact_rental和维度表。所有维度表的主键列都以<维度名称>_key来命名, <维度名称>就是维度表的表名除了dim_前缀之外的剩余部分。

在讨论sakila数据库模式的设计规范时, 曾提到源模式中的每个表里都有last_update字段, 该字段保存了一个时间戳TIMESTAMP, 用来存储每一行的最后修改 (或添加) 时间。在以后的章节中将会看到, 这个字段对于变更数据捕获非常有用, 变更数据捕获对数据持续增长的场景非常有用。虽然变更数据捕获的方法有多种, 我们这里使用的是一种最直接的方法: 每个维度表都有last_update字段, 这个字段保存了原始sakila模式中对对应表的last_update字段的值。这样可以在维度表上执行一个查询, 获得最后加载日期/时间, 并用这个日期/时间来识别和抽取所有源数据库里对应表的最新变更的数据行。

除了代理主键, 每个维度表也包含一列用来存储来自sakila数据模型的主键值, 例如, 星型模型中的表dim_film有一列film_id用来保存表film中的film_id, 当你读完后面的内容, 就会知道这些列的重要性, 这些列用来判断变更的数据是增加的还是更新的数据。

安装租赁业务的星型模型

读者可以通过本书的网址下载租赁星型模式的SQL脚本, 类似于sakila示例数据库, 这些脚本同样是.zip和.tar.gz形式的压缩包。

租赁星型模型的安装步骤同源sakila示例数据库的相同, 将压缩包解压后在MySQL命令中使用SOURCE命令执行脚本。唯一不同的是脚本文件名。在租赁星型模型中, 脚本文件名分别是

sakila_dwh_schema.sql和sakila_dwh_data.sql。

有一点需要注意的是，在后面描述的ETL解决方案中，将把 sakila 源数据库中的数据加载到星型模型中，所以这里我们只导入星型模型的表结构即可，不用加载数据，如果使用加载数据的SQL脚本加载了数据，后面的ETL流程也能执行：只不过不能再次加载变化的数据。

4.2 预备知识和一些基础的Spoon技巧

接下来详细介绍这个例子的ETL解决方案。我们建议读者自己运行这些示例。在查看这些转换和作业之前，要确认这些文件可以打开。

此外，为了使不熟悉 Spoon的读者获得一些使用 Spoon的技巧，可以参考第2章的“可视化编程”章节，下面列出了一些基本的操作，Spoon的图形界面很简单。我们相信读者可以在使用Spoon的过程中，自己解决很多问题。

4.2.1 安装ETL解决方案

读者可以从本书网站中下载的源码中找到Chapter 4文件夹中的ch4_ktr_and_kjb_files压缩包，下载并解压这些文件。

创建数据库用户

ETL解决方案中的转换使用了两个数据库用户去访问sakila数据库和租赁星型模型：第一个是sakila用户，用来连接 sakila示例数据库。另一个是sakila_dwh用户，用来连接租赁星型模型数据库。使用mysql的命令行客户端登录MySQL，使用超级用户，例如root账户进行登录，然后输入下列命令，可以创建这两个用户：

```
CREATE USER sakila IDENTIFIED BY 'sakila';
GRANT ALL PRIVILEGES ON sakila.* TO sakila;

CREATE USER sakila_dwh IDENTIFIED BY 'sakila_dwh';
GRANT ALL PRIVILEGES ON sakila_dwh.* TO sakila_dwh;
```

为了操作方便，上述代码存放在压缩文件中一个名为create_sakila_accounts.sql的SQL脚本中。

4.2.2 Spoon使用

我们假设读者已经安装了Kettle 并且可以成功地运行 Spoon 程序，安装 Kettle和运行 Spoon 命令的说明详见第3章。

打开转换和作业文件

使用 Spoon主菜单下的“文件”→“打开”菜单命令，可以选择打开文件，这时会打开一个文件选择框，在这里选择转换和作业文件的解压目录。

打开步骤配置对话框

之前我们说过本章不会对 ETL 解决方案里的每个步骤都做详细说明，读者可以根据书中的内容仔细查看 Spoon 里的转换。如果对一些转换步骤有兴趣的话，可以双击任何一个步骤，将会显示出这个步骤的详细配置。

不必担心无法理解对话框里的内容。下面的部分会结合实例详细讲解各种步骤的使用方法。

检查数据流

Kettle 的转换就是处理数据流，因此，通过查看每个步骤的输入输出流，可以深入理解转换步骤的效果，右键单击当前步骤，在右键菜单中选择“显示输入字段”或者“显示输出字段”，将会打开一个对话框来显示输入或输出流中的字段名称、数据类型、数据格式和字段来源等信息，如图 4-3 所示。

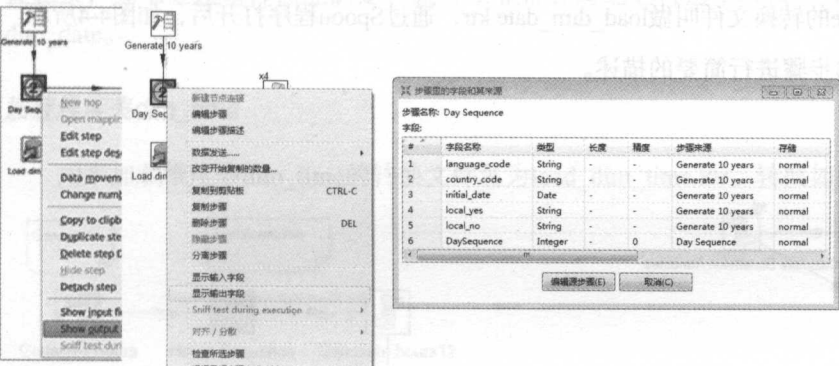


图 4-3 检查输入和输出流中的字段

运行作业和转换

如果需要运行作业或者转换，在工具栏找到“运行”按钮，即一个绿色的右箭头，在工作区的右上工具栏中的第一个按钮，单击后会弹出一个对话框，用来对运行的作业或转换进行各种配置，可以直接单击启动按钮。

当转换开始运行后，观察 Spoon 工作区下方的执行结果面板，即上一章图 3-2 右下角的区域，在这个面板中，“Step Metrics” 标签用来观察步骤执行的过程和执行速度，这是一个用来解决性能问题非常有用的工具。“Logging” 标签提供了调试和解决错误的工具，这些标签将在第 12 章和第 15 章进行详细讲解。

在执行转换的过程中，运行按钮不能单击，旁边的暂停和停止按钮可以用来暂停或者停止正在运行的转换，目前可以忽略这些按钮，只有当转换一直执行并且停不下来的时候再去使用。

4.3 ETL 示例解决方案

之前我们已经对数据库模型进行了讨论。接下来将介绍如何使用 ETL 解决方案把数据从 sakila 示例数据库加载到租赁星型模型。在剩余的部分，将分别描述把数据加载到租赁星型模型

中所使用到的作业和转换。

4.3.1 生成静态维度

维度表dim_date和dim_time属于静态维度类型：通过数据集进行初始化并且不需要定期从sakila示例数据库进行加载（虽然可能在未来需要为表dim_date生成更多的日期数据）。

第5章中将介绍，在Kimball的ETL理论体系中，类似于时间和日期的这种维度都属于“特殊维度管理”领域。虽然从术语上看可能有一些特殊工具来“管理特殊维度”，但实际上，这只是一个概念框架，那些不能直接从源系统派生来的维度都可以归到此类。在Kettle中，加载时间和日期维度并没有什么区别，可以非常容易地建立转换来生成任意日期时间数据，并分别加载到对应的维度表中。

加载维度表dim_date

加载维度表dim_date的转换文件叫做load_dim_date.ktr，通过Spoon程序打开后，如图4-4所示。接下来对图4-4中的步骤进行简要的描述。

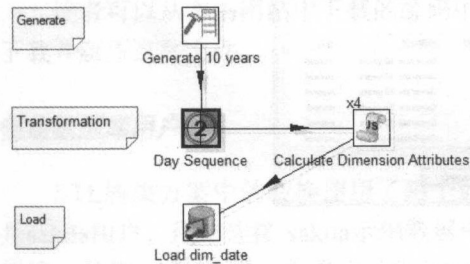


图4-4 load_dim_date转换

Generate 10 years

转换首先使用“Generate 10 years”步骤生成10年的数据行（10*366=3660），生成数据同时生成了一些常量，其中的一个常量是初始日期，设置为2000-01-01，其他的常量有语言和地区代码。

Day Sequence

图4-4中，生成的数据流入步骤Day Sequence，这个步骤生成序列，目的是为每一个输入行生成一个自增的序列数字，在后面的步骤中，这个序列号将会和initial date相加生成一系列连续的日期数据。

Calculate Dimension Attributes

图4-4中的步骤Calculate Dimension Attributes是这个转换中最重要的部分。这个步骤是JavaScript脚本，在步骤中，步骤Day Sequence作为输入流，其字段被设置为JavaScript变量，用来计算不同的日期。

步骤Calculate Dimension Attributes首先将序列号和initial date相加，生成日历对象，然后，通过JavaScript表达式将其转换为不同的日期和日期的各个部分。同时，还使用JavaScript表达式生成智能主键，用于区别表dim_date中的数据。

JavaScript表达式使用了语言和国家地区代码（在步骤“Generate 20 years”中设置）来完成对日期做本地化设置。例如，国家代码是gb且语言代码为en，日期2009-03-09被格式化为Monday, March 9, 2009，如果国家地域代码是ca且国家代码是fr，那么同样的日期将会被格式化为lundi 9 mars 2009。

步骤Calculate Dimension Attributes还阐述了两个非常有趣的特性，在图标的左上角，有一个“x4”的标签，这个图标并不是这个步骤的一部分，而是表示此步骤生成的拷贝数量，可以右键单击此步骤，然后在菜单中选择“改变开始复制的数量”。更多细节详见第15和16章。

Load dim_date

转换中最后一个步骤是Load dim_date，这是一个表输出步骤，接收Calculate Dimension Attributes步骤输出的数据，然后生成相应的SQL命令，最终将数据插入维度表dim_date中。

运行转换

运行转换的前提条件包括数据库已经正常运行，安装了sakila_dwh数据脚本，用户账户设置无误，如果上述条件已经满足，就可以非常容易地运行这个转换，几秒之内将数据加载到dim_date。

加载维度表dim_time

用来加载维度表dim_time的转换文件名为load_dim_time.ktr。转换如图4-5所示。

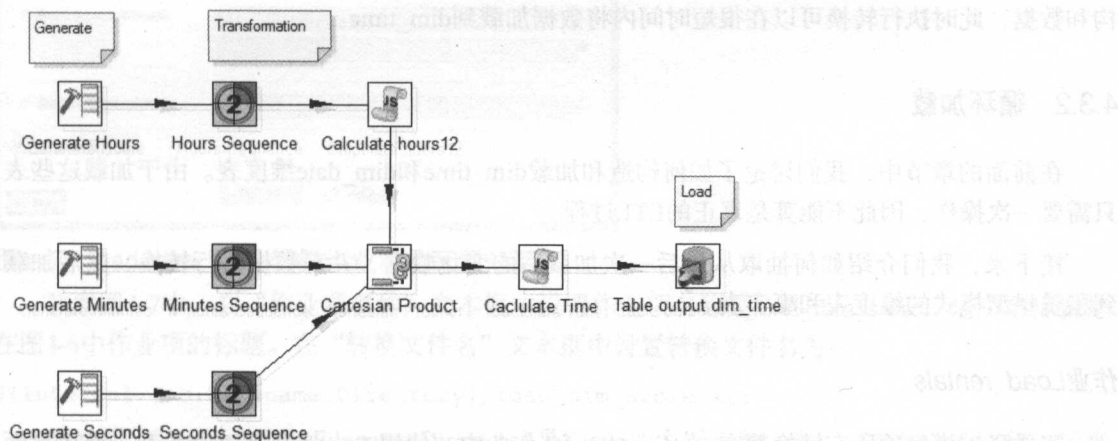


图4-5 load_dim_time转换

转换load_dim_time中使用到的步骤，不仅包括已经介绍了的“生成记录行”步骤、“增加序列”步骤、“Java脚本”步骤和“表输出”步骤，还包括了“记录关联（笛卡儿输出）”步骤。接下来讨论该转换中的关键步骤。

生成小时、分钟、秒

和转换load_dim_date类似，转换load_dim_time也通过“生成行”步骤初始化数据，但是这个转换中有3个生成行步骤并行执行：

- 步骤Generate Hours生成24行数据，用来表示一天中的24小时。
- 步骤Generate Minutes生成60行数据，用来表示每小时的60分钟。
- 步骤Generate Seconds也生成60行数据，用来表示每分钟的60秒。

每一个生成行类型的步骤在随后的序列类型步骤中获得一系列连续整数0 ~ 23小时、0 ~ 59分钟、0 ~ 59秒，其中在小时的分支中，在序列步骤之后的步骤为Calculate hours12，这是一个JavaScript脚本步骤，用来获得12小时制的表示方法，并通过计算生成12小时制的符号AM/PM。

笛卡儿乘积

接下来是一个叫做“Cartesian Product”的步骤。这是一个“记录关联（笛卡儿输出）”步骤，用来连接两个输入流，生成笛卡儿乘积输出流，也就是两个输入流数据行的各种组合方式（m*n），每一个输出行包含所有输入行的字段。

尽管通过限制“记录关联（笛卡儿输出）”步骤，可以只生成特定组合行，但是在这里我们需要生成所有的组合。一共生成24*60*60=84600行，每一行代表24小时制中的一秒（实际上并不完全正确，因为没有考虑闰秒。我们忽略这个小问题继续往下看）。

Calculate Time和Table out dim_time

“记录关联（笛卡儿输出）”步骤之后是一个“JavaScript脚本”步骤和一个“表输出”步骤，这些步骤的功能与load_dim_date中的最后两步基本相同：输入流中的值通过JavaScript表达式进行修改后生成需要的时间格式，另外生成 dim_time 维度表的智能主键，最后将数据插入到维度表。

运行转换

和运行load_dim_date转换相同，假设数据库处于运行状态，并且已经安装了sakila_dwh表结构和数据，此时执行转换可以在很短时间内将数据加载到dim_time。

4.3.2 循环加载

在前面的章节中，我们讨论了如何构造和加载dim_time和dim_date维度表。由于加载这些表只需要一次操作，因此不能算是真正的ETL过程。

接下来，我们介绍如何抽取从最后一次加载后的变化数据，并对数据进行转换、然后加载到租赁星型模式的维度表和事实表。

作业Load_rentals

租赁星型模型的所有转换都在 load_rentals.kjb作业中。使用 Spoon 打开这个作业，整个界面如图4-6所示。

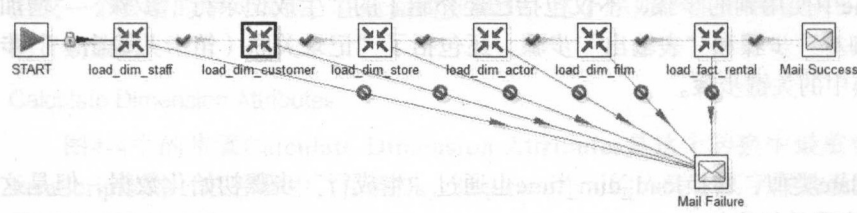


图4-6 load_rentals作业

接下来将讨论作业里的关键内容。

Start

图4-6中作业 load_rentals的第一个作业项是START节点。每个有效的作业都只能有一个START节点，用来表明作业开始运行的位置。

不同的转换作业项

START作业项连接到后面的转换作业项。位于START作业项和第一个转换作业项之间的连接线上有一个锁标记，这个符号表明这是一个无条件的连接，即使之前的作业项没有执行成功也会执行下一个作业项。在下面的章节中，将讲述其他的连接类型。

START后面的各个转换作业项用来运行不同的转换，完成特定的功能。例如：load_dim_staff作业项执行load_dim_staff转换（用来加载dim_staff表），紧随其后的load_dim_customer作业项执行load_dim_customer.ktr转换（用来加载dim_customer表），等等。

为了进一步了解转换是如何同转换作业项关联的，双击转换作业项。会弹出转换设置对话框，在这里配置转换作业项。例如，双击load_dim_store作业项（图4-6的第三个转换），将会看到如图4-7所示的配置对话框。

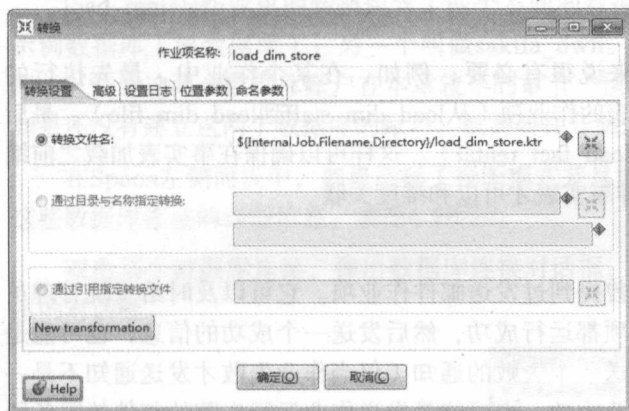


图4-7 load_dim_store转换作业项的配置对话框

注意图4-7中，在“作业项名称”文本框中设置作业项名称是load_dim_store。这个名称就是在图4-6中作业项的标题。在“转换文件名”文本框中设置转换文件名为：

```
${Internal.Job.Filename.Directory}/load_dim_store.ktr
```

前半部分`${Internal.Job.Filename.Directory}`是Kettle的内置变量，表示当前作业的路径。后半部分`/load_dim_store.ktr`是转换文件的名称。

注意：变量非常重要，它可以使转换和作业中类似文件或服务器这样的资源更加独立。图4-7就是一个非常好的例子：它替代了在作业项中对文件路径的硬编码，对当前的作业来说`${Internal.Job.Filename.Directory}`这个内置变量的路径是一个特定的相对路径。

有关变量的话题已经在第3章简单讨论过，在本书中会有更多的地方讨论。

对话框中“转换文件名”和“作业项名称”没有关系，可以使用任何名称来命名作业项。load_rentals例子中作业项的名称和转换的文件名相同，这种一样的命名方式可以使作业项清晰地关联到转换文件。

如果需要浏览当前的转换文件，右键单击转换作业项并从菜单中选择“Open transformation”

选项。在本章的学习过程中, load_rentals作业可以一直打开, 这样可以方便地浏览所有转换。

不同类型的作业项连接和流程异常

在 load_rentals 作业中, 每个转换作业项后面都有一个执行成功的作业项连接: 执行成功的作业项连接是绿色的并且指向下一个作业项, 每个转换作业项还有一个执行失败的作业项连接; 执行失败的作业项连接是红色的(带一个停止图标)并且指向一个发送邮件作业项。

当运行这个作业时, 作业项将按顺序执行: 首先, 启动load_dim_staff作业项并执行相关的转换, 如果执行成功, 作业将会在连接线上显示成功标志, 然后继续执行作业项load_dim_customer, 然后依次执行其他转换作业项, 依此类推, 直到最后一个转换被成功执行, 最后执行发送邮件作业项Mail Success, 作业执行成功。

当然, 也有一种可能是有某个作业项执行失败, 此时, 作业将会显示是哪个作业项执行失败, 然后继续执行Mail Failure 作业项, 最后整个作业停止执行。

顺序执行

注意作业里的作业项是顺序执行的, 这和转换不同: 在转换中, 所有的步骤同时启动运行。数据流从输入步骤顺序流入, 然后从输出步骤流出。

作业项的顺序执行对于同步的工作来说很有必要, 例如, 在某个作业中, 最先执行的START 作业项后面有一系列用来加载维度表的作业项(从load_dim_staff到load_dim_file), 最后才是加载事实表fact_rental的转换作业项(load_fact_rental)。这样可以确保在事实表加载之前维度表中新的维度行已经全部加载完毕, 这样事实表才可以和维度关联。

邮件作业项

前面在讨论不同类型的作业项连接时曾提到过发送邮件作业项。它可以及时给系统管理员报告作业的运行状态: 可能是所有的作业项都运行成功, 然后发送一个成功的信息, 也可能是某个作业项运行失败, 作业提前结束并发送一个失败的通知(仅当作业失败才发送通知不是一个很好的办法, 没有收到邮件并不代表作业成功, 也可能是发送作业运行失败的邮件的服务器发生故障)。

在使用邮件作业项时, 需要配置邮件服务器, 双击作业项, 然后在“服务器”标签页的位置填写合适的邮件服务器地址。如果你不能确定如何填写, 可以暂时不用设置发送邮件作业项, 这个加载租赁星型模型的作业也可以不用配置发送邮件作业项。只是不能收到关于作业执行状态的E-mail的通知。

运行作业

运行作业的方式同运行转换完全相同。运行作业项可能需要一点时间, 但是应该会在一分钟内结束。不用担心再重新运行一遍作业或某个单独的转换, 如果租赁星型模型已经是最新的, 作业就会发现没有要处理的数据。

load_dim_staff 转换

作业load_rentals中第一个要执行的转换是load_dim_staff, 如图4-8所示。

这个转换的目的是加载dim_staff维度表。接下来讨论这个转换如何完成这项工作。

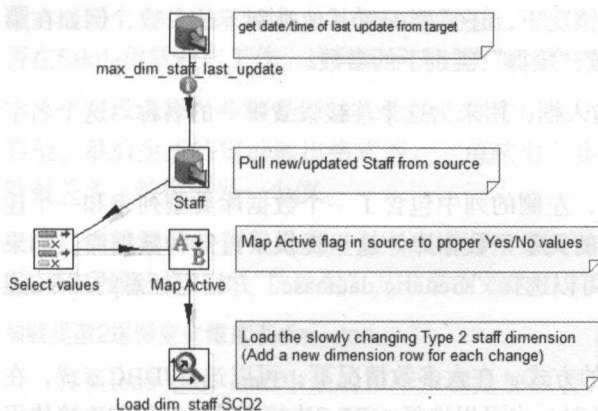


图4-8 load_dim_staff转换

数据库连接

load_rentals作业里的转换包含了两个不同的数据库连接：一个叫做sakila，用来连接sakila示例数据库（源数据库）；另一个叫做sakila_dwh。用来连接租赁星型模型数据库（目标数据库）。注意这两个数据库账户在本章较早的章节“预备知识和一些基础的Spoon技巧”中建立，如果还没有建立这两个数据库的账户，就需要在开始之前重新阅读“创建数据库账户”。

在Spoon左侧面板中，如果选择了视图模式并且展开“数据库连接”的文件夹，就可以查看这些数据库连接的详细信息，如图4-9所示。

双击某个数据库连接，弹出数据库连接对话框，例如，双击sakila连接将会弹出如图4-10所示的对话框。

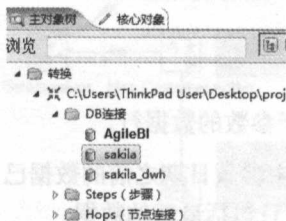


图4-9 左侧的数据库连接面板

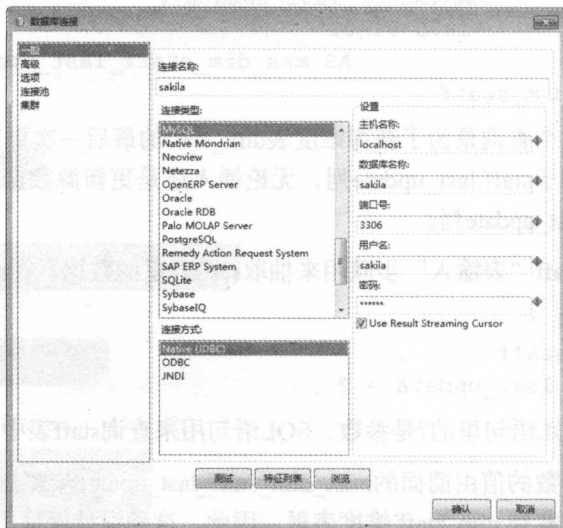


图4-10 sakila连接的数据库连接对话框

在Kettle中这是一个经常使用的对话框，有很多设置内容，下面看在这个数据库连接对话框里可以设置哪些参数。

在对话框的左侧，是参数的类别，一般情况下，数据库配置在“一般”类别下（如图4-10中

选择的分类)就可以完成,但是在一些特殊情况下,还需要配置其他类别下的参数,例如在第17章中,为了配置一个数据库集群,还要设置“集群”类别下的参数。

在对话框的右侧,首先是“连接名称”输入框,用来为这个连接设置唯一的名称。这个名字将会在作业或者转换中使用。

在“连接名称”输入框下方有两列信息,左侧的列中包含了一个数据库类型列表和一个连接方式列表。从数据库类型列表中选择需要的关系型数据库。这里提供了近30种数据库,如果你发现你的数据库不在这30种数据库之内,可以选择“Generic database”并指定任意的JDBC连接串。

在连接方式列表里可以选择连接数据库的方式,在大多数情况下,可以选择JDBC方式,在少数情况下,如不能得到数据库的JDBC 驱动时,也可以选择 ODBC连接方式。ODBC连接使用Sun的JDBC/ODBC桥。

注意:除了JDBC和ODBC之外,还可以使用JNDI连接方式,JNDI连接从技术上来说也是JDBC连接,但是使用 Java命名和目录接口的方式定义。第3章更详细介绍了JNDI的配置方法。

对话框右侧需要设置一些建立数据库连接时必要的参数,设置这些参数依赖于数据库类型和连接方式类型,因此很难对其进行定义。

变更数据捕获与抽取

Load_dim_staff转换中前两个步骤都是“表输入”步骤,这个步骤可以执行一段SQL语句并且将返回的数据行转换为转换的输入流。

max_dim_staff_last_update步骤使用sakila_dwh数据库连接执行下面的查询:

```
SELECT COALESCE(
    MAX(staff_last_update),
    '1970-01-01'
) AS max_dim_staff_last_update
FROM dim_staff
```

这个查询是为了获得维度表dim_staff的最后一次更新时间, dim_staff.staff_last_update字段的值来源于staff.last_update列,无论插入还是更新源数据库的staff表,当前的操作时间都会自动保存在last_update列。

Staff“表输入”步骤用来抽取staff表里的数据,在sakila数据库中执行下面的查询:

```
SELECT *
FROM staff
WHERE last_update > ?
```

SQL语句里的?是参数,SQL语句用来查询staff表中last_update列大于参数的数据行。

参数的值由前面的max_dim_staff_last_update步骤提供,由于表staff中参数日期之前的数据已经被加载到 dim_staff 维度表里。因此,这种设计保证了Staff步骤的输出只包括最新的数据。

在任何ETL解决方案中变更数据捕获都是一个重要的话题,被认为是ETL子系统2,关于变更数据捕获的更多内容将在第5、6章讨论。

转换和映射Active标记

转换中接下来的两个步骤是“Select values (字段选择)”和“Value Mapper (值映射)”步

骤，这两个步骤用来转换一个布尔类型的值，把这个值转化为Yes/No值，这个值表示员工当前是否在Sakila租赁商店工作。这种表示方式更加适合报表和分析。

“字段选择”步骤是对输入流的一种常用操作。例如，删除列、重名列、转换列的数据类型，最后生成特定的输出格式等。“值映射”步骤，顾名思义，把输入流中指定的值，通过映射关系，转换成另一个值。

数据转换和映射是数据一致性的例子，属于ETL子系统8，在第5章和第9章有更详细的描述。“字段选择”和“值映射”是非常有效的用来做数据转换的步骤。

加载类型2缓慢变化维度表dim_staff

转换load_dim_staff中最后一个步骤是加载表dim_staff。这个表是一个类型2的缓慢变化维度：源数据库中数据行的任何变化都会使目标数据库中的维度表中增加一行记录，新增加的一行是原记录的一个版本。

在维度表中，一个职员记录有不同的版本，所有这些版本的staff_id都是相同的，staff_id是原表中职员的id。另外，维度表还有一对列：staff_valid_from和staff_valid_through，用来说明每个职员版本适用的时间。此外还有列staff_version_number，用来维护这行数据的版本号。

缓慢变化型维度的管理被称为ETL子系统9（缓慢变化维度处理），第5章将会有更深的讲解。Kettle提供了一个专用步骤“维度更新查询”，用来维护类型2的缓慢变化维度。在第8章将详细介绍如何使用这个步骤。

load_dim_customer转换

load_dim_customer转换是load_rentals作业里的第二个作业项。目的是加载维度dim_customer，转换如图4-11所示。

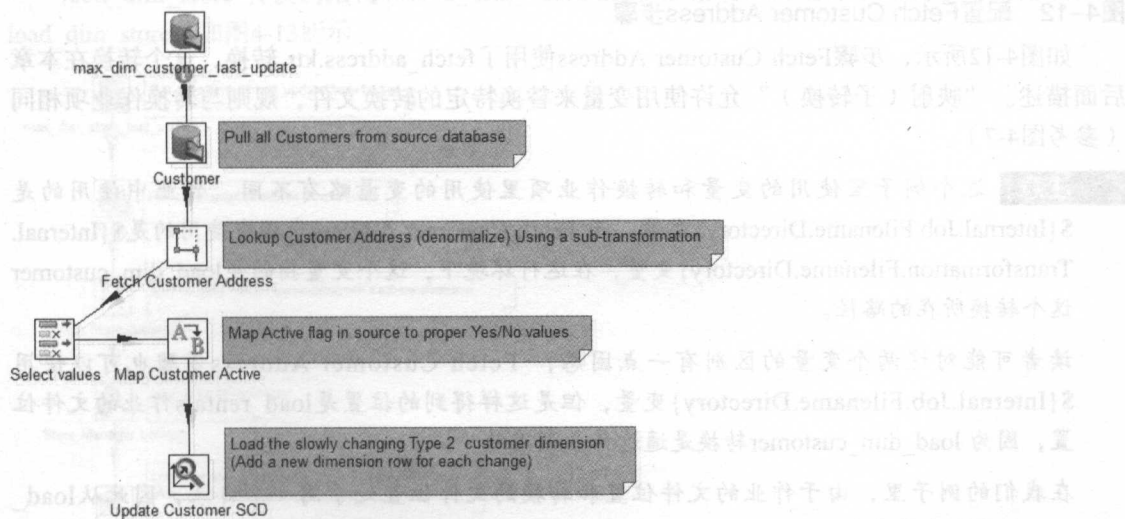


图4-11 load_dim_customer转换

load_dim_customer转换的结构同之前的load_dim_staff转换：

- 转换开始的地方是两个“表输入”步骤，用来捕获并抽取变更的数据。
- 转换结束的地方使用“维度查询/更新”步骤来维护类型2的缓慢变化客户维度，并把数

据加载到维度表dim_customer。

- 数据在加载到维度表dim_customer之前，表customer中的active列的值被转换为Yes或者No。

Fetch Customer Address子转换

除了和load_dim_staff 转换相同的步骤外， load_dim_customer 转换还包含一个特殊的步骤叫 Fetch Customer Address，这是一个映射（子转换）步骤，允许重复调用一个存在的转换。

如图4-12，双击步骤，可以打开配置窗口，查看 “映射（子转换）” 步骤的设置。另外当右键单击这个步骤并且从菜单中选择 “打开映射（子转换）” 选项时，可以直接打开这个子转换。

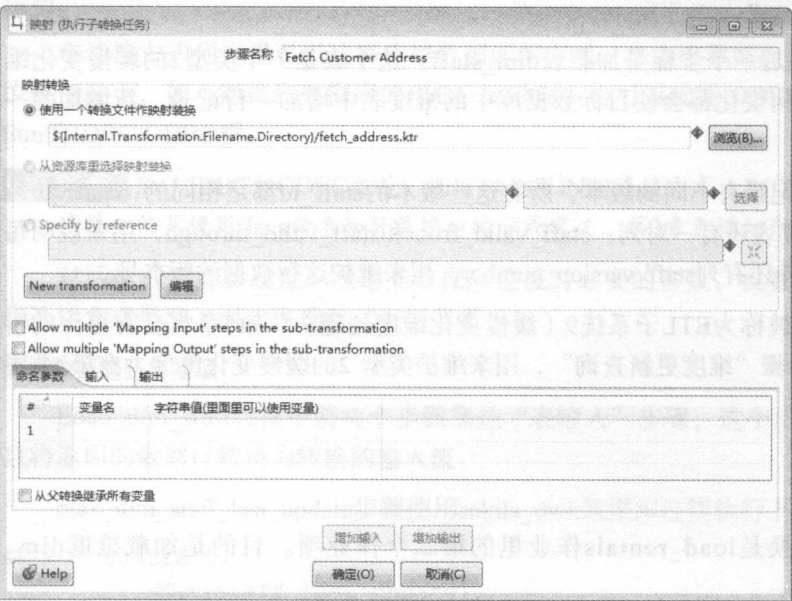


图4-12 配置Fetch Customer Address步骤

如图4-12所示，步骤Fetch Customer Address使用了fetch_address.ktr 转换，这个转换在本章后面描述。“映射（子转换）” 允许使用变量来替换特定的转换文件，规则与转换作业项相同（参考图4-7）。

注意：这个例子里使用的变量和转换作业项里使用的变量略有不同，作业中使用的是 `$(Internal.Job.Filename.Directory)` 变量，而 Fetch Customer Address 步骤中使用的是 `$(Internal.Transformation.Filename.Directory)` 变量，在运行环境下，这个变量指的是load_dim_customer 这个转换所在的路径。

读者可能对这两个变量的区别有一点困惑， Fetch Customer Address 步骤也可以使用 `$(Internal.Job.Filename.Directory)` 变量，但是这样得到的位置是load_rentals作业的文件位置，因为 load_dim_customer转换是通过作业调用的。

在我们的例子里，由于作业的文件位置和转换的文件位置处于同一个目录，因此从load_rentals作业中调用 load_dim_customer转换时，这两个变量的取值没有什么区别，但是这里必须明确Internal.Job.Filename.Directory和Internal.Transformation.Filename.Directory并不一定是同一个路径。假如直接运行转换load_dim_customers（不通过作业调用），变量Internal.Job.Filename.Directory由于没有初始化而不可用，然而Internal.Transformation.Filename.Directory 变量仍然指当前转换文件的路径。

从功能上说，Fetch Customer Address步骤的目的是为了查找客户的地址，这种做法对于反正规化的维度表dim_customer是非常有必要的，维度表里的数据来自 sakila.address表以及和它相关的sakila.city、sakila.country表。dim_store 维度表同样也是反正规化设计的，也需要引用地址数据，所以我们要把获取地址的转换作为子转换，供不同转换使用。

注意：“映射（子转换）”步骤同一些程序语言中把某个代码源文件导入到主程序中有些类似，由于创建和测试一块逻辑代码需要耗费许多时间，因此任何需求变更或者bug修复都应该在一个模块当中，以提高系统的复用性。

通常情况下，与重复开发相同的转换相比，使用子转换可以节省大量的开发和维护时间。但是客观地说，重用也有一定成本，因为要耗费更多的精力为重用组件设计合理的接口，并且当修改重用接口时可能会影响当前ETL解决方案中其他用到此组件的地方。

在这个租赁星型模型中，你可能会觉得不应该多次查询地址：如果星型模型在设计上再宽松一点，并且把地址作为一个单独的维度表来维护，表dim_customer和dim_store可能就会成为雪花型维度，地址维度表实际就是Kimball所说的支架维度表，就是把所有维度表里的地址信息，作为一个单独的地址维度表分离出来。其实这是另一种形式的重用，是在数据仓库层次上而不是在 ETL 层次上。

是否应该做这种正规化或反正规化，即这个例子里是否应该把地址单独作为一个维度表分离出来，这个问题是BI和数据库专家们争辩的导火线，在这一点上我们并不想站在任何一边。但是，我们要利用这个机会来说明，Kettle可以复用转换，以应对重复的业务逻辑。是否要使用工具来解决这个问题，最终还是要读者自己来回答。

load_dim_store 转换

load_dim_store 转换的结构和load_dim_customer非常相似，就是为了加载缓慢变化维度表load_dim_store，如图4-13所示。

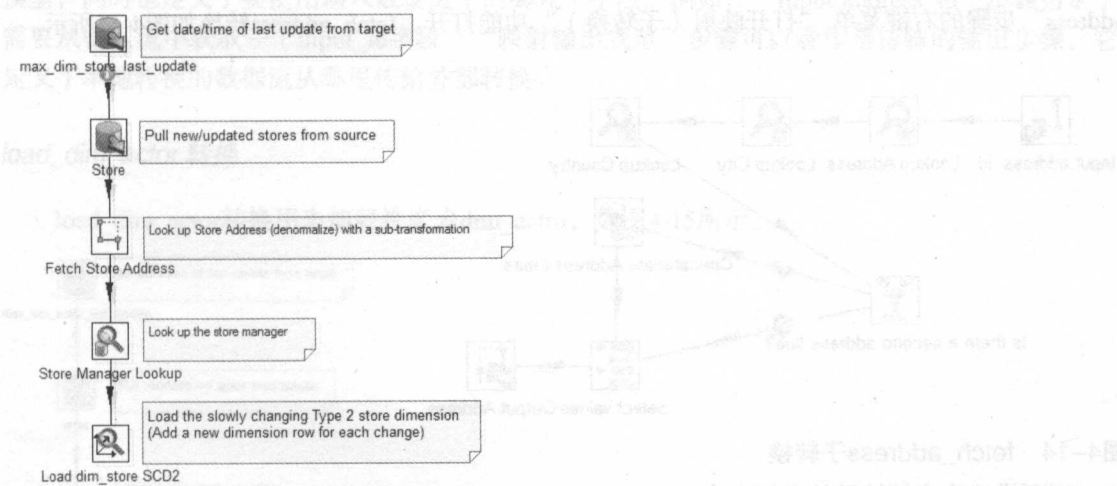


图4-13 load_dim_store转换

在该转换中，变更数据捕获的原理与load_dim_staff转换和load_dim_customer转换的原理基本相同，它同样使用“维度更新查询”步骤来加载并维护维度历史信息。区别在于数据源表和目标表变成了store和dim_store。和load_dim_customer转换类似，转换load_dim_store也使用了

“映射（子转换）”步骤来调用 `fetch_address` 转换以取得商店的地址。

`load_dim_store` 转换中有一个之前没遇到的步骤：“Database lookup（数据库查询）”步骤。该步骤名称是“Store Manager Lookup”，该步骤使用SQL语句从数据库中查询数据。SQL语句使用前面步骤传来的数据作为SQL参数，这些参数一般是数据库中的主键或者唯一约束，该步骤的输出列包括了输入列的所有字段，加上在查询中新增的字段，该步骤一般有下面几个用途。

- 数据清洗和一致性：和值映射步骤类似，数据库查询步骤可以用于数据清洗和一致性。它们之间的区别在于值映射步骤中查询的值是作为常量录入的，而数据库查询步骤的值来源于关系数据库。
- 反规范化：类似于SQL语句中的join连接符把不同表的列连接起来，“数据库查询”可以把输入流中的列和数据库查询的列连接起来形成新的输出。
- 维度表查询：当加载事实表时，数据库查询步骤可以查询维度表的主键值。

注意：Kettle提供了其他的查询步骤，比如“维度查询/更新”步骤和“组合查询/更新”步骤，这些步骤提供了比“数据库查询”步骤更多的查询功能，一般用于维护类型2的缓慢变化维度或者杂项维度，这些步骤将在第8章中详细介绍。

数据库查询步骤用途很广，可以用于第5章中描述的多个ETL子系统：子系统8（数据一致性）、子系统14（代理键管道）和子系统17（维度管理系统）。

从功能上，`load_dim_store` 转换里的“Store Manager Lookup”数据库查询步骤用于反规范化源数据库里的store表，以便可以加载到 `dim_store` 维度表中，这里的“数据库查询”步骤属于维度管理系统（子系统17）。

`fetch_address` 子转换

前面讲过`load_dim_customer`转换和`load_dim_store`转换使用“映射（子转换）”步骤调用`fetch_address`转换。你可以直接打开`fetch_address.ktr`文件，或者通过转换里的“Fetch Customer address”步骤的右键菜单“打开映射（子转换）”功能打开。`fetch_address`转换如图4-14所示。

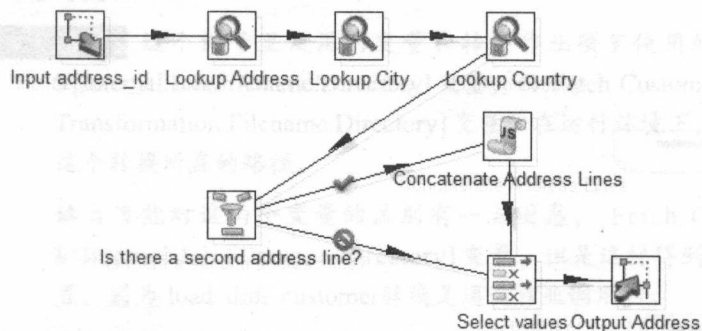


图4-14 `fetch_address`子转换

下面描述这个转换里的主要内容。

地址数据级联查询

转换`fetch_address`中包括了一系列的“数据库查询”步骤，这些步骤包括Lookup Address、Lookup City和Lookup Country。这些步骤形成了级联查询：Lookup Address步骤使用输入流的

address_id字段在address表中查询到对应的数据行，然后作为查询结果字段之一的city_id增加到输出流，接下来，Lookup City步骤使用这个字段在表city中获得对应的数据行，并把country_id增加到输出流，最后，使用 country_id作为查询条件从表country中获得对应的数据行，至此，输出流中已经获得了与输入流中address_id对应的所有地址数据。

更多的反正规化：拼接地址信息

在sakila数据库中，address表用两列address和address2来存放多行地址，但是dim_customer维度表和dim_store维度表只有一个地址字段，所以必须要处理多行地址。

fetch_address转换里有一个过滤步骤，名为“Is there a second address line?”。这个步骤将地址输入流分为两个输出流：一个数据流是多行地址的情况，另一个数据流是单行地址的情况。

在图4-14中，多行地址是“过滤”步骤的“true”输出，对应的连接线上有一个对钩号标记。后面再使用Java脚本步骤处理这些地址：把多行地址合成一个字段。

无论是多行地址还是单行地址，数据最终都流向“字段选择”步骤，在这个转换中，“字段选择”步骤只选择输出需要的字段，丢弃了所有类似于city_id和country_id这样的中间字段。

虽然也可以输出 fetch_address 转换里产生的所有字段，但不建议这样做，因为这样所有调用这个子转换的转换不得不计算哪些字段是有用的哪些是没用的。另外，一个外部转换可能有意无意地使用了子转换输出的某些字段。所以在维护fetch_address子转换时，需要确认这些字段是否被使用。

子转换接口

fetch_address转换开始和结束步骤分别是“映射输入规范”（步骤名称“Input address_id”）和“映射输出规范”（步骤名称“Output Address”）步骤。任何子转换都需要这两个步骤，它们是子转换的接口。

“映射输入规范”步骤可以看作是特殊的输入步骤，该步骤把外部转换的数据注入到子转换里，同时也定义了要使用输入数据流中的哪几个字段。例如，“Input address_id”步骤指定了需要从输入流中获取一个input_id字段。“映射输出规范”步骤可以看作是特殊的输出步骤，它定义了本地转换的数据流从哪里传给外部转换。

load_dim_actor 转换

load_dim_actor转换用来加载维度表dim_actor，如图4-15所示。

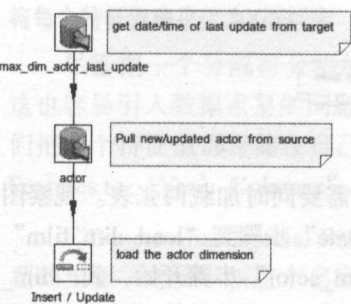


图4-15 load_dim_actor转换

这是目前为止最简单的一个转换，它使用了和之前所有转换相同的变更数据捕获方式。这

里介绍一个新的转换步骤：“Insert/Update（插入/更新）”。

插入/更新步骤

load_dim_actor转换使用“插入/更新”步骤把数据加载到表dim_actor。该步骤的工作原理是使用数据库表的关键字比较输入流的关键字，如果数据流里的关键字在数据库中已经存在，程序将更新数据库里该关键字数据行里的某些指定字段。如果这行数据不存在，程序将把数据流里的这行数据写入到数据库中。

对于 dim_actor 维度表，“插入/更新”步骤已经足够了，因为dim_actor维度表并不是一个缓慢变化维度（至少不是类型2的缓慢变化维度），所以我们只做更新就可以。例如，演员名字写错了，直接修改就可以。

load_dim_film 转换

load_dim_film转换用来加载dim_film维度表和dim_film_actor_bridge连接表，如图4-16所示。

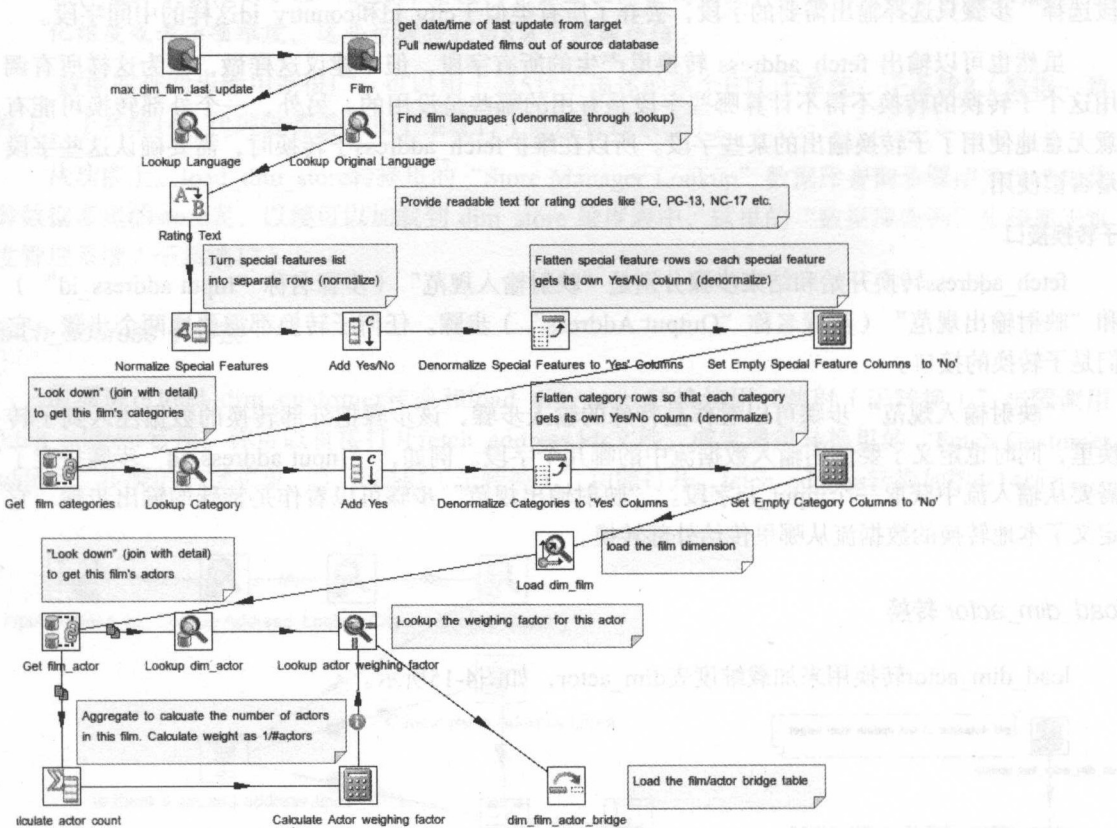


图4-16 load_dim_film转换

load_dim_film转换是我们目前遇到的最复杂的转换，因为它需要同时加载两张表。观察图4-16，可以发现整个转换的前半部分是从“max_dim_film_last_update”步骤到“load_dim_film”步骤。这部分主要负责加载维度表dim_film。下面部分从“Get film_actor”步骤开始，到“dim_film_actor_bridge”步骤结束，用来加载dim_film_actor_bridge表。另一个导致转换比较复杂的原因是需要解决dim_film表的几个多值维度问题。最后，表dim_film本身还要处理非正规化问题，例如，原表引用了语言表，要通过数据库查询做反正规化处理。

首先看load_dim_film转换中使用到的和以前转换相同的步骤。

- 使用两个“表输入”步骤max_film_last_update和Film来捕获变化的数据。
- 两个“数据库查询”步骤：Lookup Language和Lookup Original Language，用来查找电影的语言。
- 使用值映射步骤：Rating Text 转换源数据库film表中的rating字段的值。

我们不深入讨论这部分转换，前面的部分已经详细描述了这些技术，下面讨论前面没介绍过的内容。

- 源数据库film表里的special_features字段是一个非原子的值列表（MySQL的SET类型），在dim_film表中，它被转化成一组字段，字段值是Y/N，字段名是film_has_%。
- 源sakila库中，存储在film_category和category表中的电影多个分类也需要扁平化、非正规化成 dim_film表中的一组字段（film_is_%）的Y/N值。

上面介绍了加载dim_film表的转换。对于dim_film_actor_bridge表，要说明两件相关但不同的事情：

- 对于添加到dim_film维度表中的每一行，必须要获取到这个电影里所有演员的列表。对于每个演员，还要从之前加载的dim_actor表中查找对应的actor_key，以便插入到dim_film_actor_bridge 表。
- 对于添加到dim_film_actor_bridge表中的数据，为了把dim_actor维度分配给fact_rental事实表的度量列，必须给每个电影里的演员计算和分配一个权重因子。

下面再详细讨论上面提到的这些转换技术。

分割special_features列表

源数据库film表中的电影特征列里的值是以逗号分隔的字符串，为了把这个字符串转换成维度表里的多个特征列，首先要将逗号分隔的字符串正规化为一组行。这样后面才能把每个电影里的多个特征值放到电影维度表的多个特征列里。

“Normalize Special Features” 步骤用来解析和分割特征值字符串。这是一个“列拆分为多行”步骤。这个步骤可以把一个列里以分隔符分割的字符串，按分隔符拆分为多个数据行。

例如，输入流里有一个special_features列表，值是“Deleted Scenes, Trailers”，那么该步骤就会在输出流中输出两行：一个是“Deleted Scenes”，另一个是“Trailers”。除了special_feature字段，这两行中的其余字段完全相同。

将每个特征值扁平化为Y/N标志

尽管把一个分隔符分割的特征字符串列表正规化为多行的形式便于处理每个特征值，但这也容易引入数据重复的问题。有时候我们也可以重新扁平化每个电影的多行数据，但这里我们把每个特征值都存储在自己的列中，而不是存储在一个列的列表中。“Denormalize Special Features to ‘Yes’ Columns” 步骤完成了这个功能。

“Denormalize Special Features to ‘Yes’ Columns” 步骤是一个“行转列（反正规化）”类型的步骤。之所以叫这个名字是因为它可以把多行数据转换为多列数据。仔细查看这个步骤的配置，可以更深入地了解这个步骤的功能（如图4-17所示）。

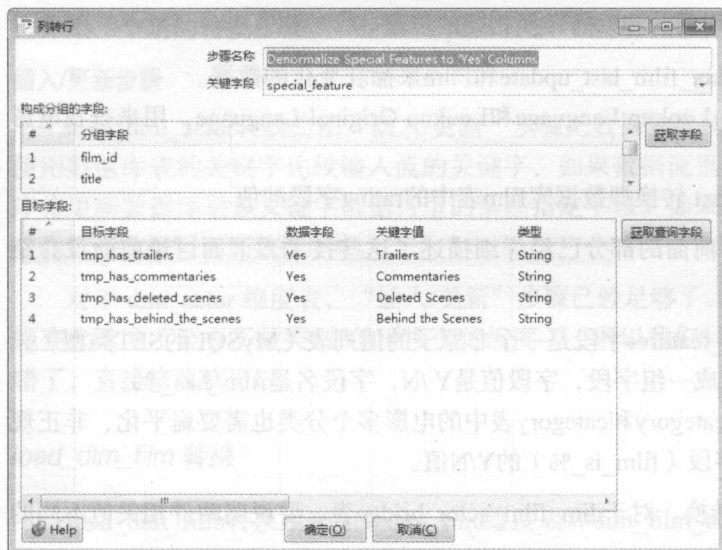


图4-17 “Denormalize Special Features to ‘Yes’ Columns” 步骤

图4-17的对话框里有三个重要的配置项如下。

- **关键字段**：此字段将被反规范化，根据这个字段生成多列。
- **构成分组的字段**：要确定一组数据所需要的字段，一般来说，这些是来自输入流的你想在输出流里保留的字段。
- **目标字段**：这里定义了要添加到输出流里的字段，用来指定一组数据里关键字段里的唯一值应该写到哪个输出字段里。例如，设置的第一行表示，当关键字段的值是Trailers时，程序向“tmp_has_trailers”字段里写入“Yes”。

注意：在“行转列”的配置对话框中，对于“目标字段”的解释稍微有些不正确。存储在新生成的值不是一个“Yes”字符串，实际它是一个字段为“Yes”的字段。图4-16里有一个“Add Yes/No”步骤（在“Denormalize Special Features to ‘Yes’ Columns”步骤前面）。这个步骤定义了两个常量字段，字段名称分别为“Yes”和“No”，相应的值也为“Yes”和“No”。

创建电影分类字段

在sakila数据库中，电影属于多个分类。film_category表存储电影分类信息，这个表里有指向film表和category表的外键。在租赁星型模型中，分类信息存储在dim_film维度表中，每个分类都是一个值为Yes/No字段。这种处理方法和上面电影特征的处理方法类似。

如图4-16所示，“Get film_categories”步骤用来获取每部电影的分类列表。该步骤是一个“数据库连接”步骤。“数据库连接”步骤与“数据库查询”步骤相似，“数据库查询”步骤，在本章前面讨论load_dim_store转换时已经介绍过。

与“数据库查询”步骤相同，“数据库连接”步骤也执行SQL查询，此SQL查询使用输入流中的数据作为参数。不同的是，在“数据库查询”步骤中，为了从数据库中查找一行，参数通常映射到一列主键或唯一约束上；在“数据库连接”步骤中，为了能检索到一组数据，参数通常映射到外键上。与“数据库查询”类似，“数据库连接”步骤类型向输出流中增加了数据库中的很多指定字段。不同的是，“数据库查询”步骤对于每一行输入流最多只能向输出流返回

一行，而“数据库连接”步骤是从数据库中返回的每一行都输出到输出流中。换句话说，“数据库连接”步骤包含了“数据库查询”的结果，同时还可能返回了更多的数据行。

创建电影分类字段与前面小节的电影特征的反正规化过程非常相似：“Get film_categories”步骤把每行电影转换成了分类不同的多行数据。后面也有一个和前面类似的行转列步骤“Denormalize Categories to ‘Yes’ Columns”，该步骤把电影的多个分类（多行数据）都转换成电影维度表里的多个列，列的值是“Yes/No”。

加载dim_film表

加载dim_film维度表是通过load_dim_film步骤完成的。这个步骤是一个“联合查询/更新”步骤，这个步骤和“维度查询/更新”步骤类似。在这一章前面的小节中，我们讨论load_dim_staff转换时曾经遇到过“维度查询/更新”步骤。

我们在第8章“处理维度表”中再详细讨论“联合查询/更新”步骤。现在，我们只要把这个步骤当作和load_dim_actor转换中的“插入/更新”步骤相类似的步骤就可以：数据若不存在，则在dim_film dimension表中插入一行；若存在，则更新该行。

与“插入/更新”步骤类型不同的是，“联合查询/更新”步骤还会返回该维度行的键。这就是为什么我们要选择这个特殊的步骤：我们需要dim_film行的键，在后面的转换中需要把它们加载到dim_film_actor_bridge表里。

加载dim_film_actor_bridge表

加载dim_film_actor_bridge表的第一步是从sakila数据库的film_actor表中获取电影的演员名称。通过“数据库连接”步骤“Get film_actor”可以得到演员名称。实际上，装载桥接表的时候，还需要获取另外两个数据。这就是为什么“Get film_actor”步骤有两个输出。

首先，需要从dim_actor表查找actor_key的值。对于“数据库查询”步骤来说，这是一个简单的任务。在load_dim_film转换中，通过“Lookup dim_actor”步骤来实现。

其次，你需要计算一个权重因子，并将它连同每个film_key / actor_key组合存储在dim_film_actor_bridge表中。因为sakila数据库中沒有可以帮助你确定每一个演员对电影实际贡献的数据，所以可以简单地假设每个演员都有相同的权重。可以通过以下公式来计算每个电影中任意演员的权重：

$$\text{权重} = 1 / \langle \text{电影中的演员数} \rangle$$

使用两个步骤可以计算这个公式：

- “Calculate actor count”步骤是“分组”类型的步骤，主要做两件事：1. 它与SQL语言中的GROUP BY子句相似，基于指定的字段（或字段集），将输入流中的数据行进行分组；2. 为整个分组生成一个或多个聚集值，发送到输出流。

在这个例子里，分组字段是film_key。此外，我们要计算演员的总量（每个film_key），它会输出到输出流的count_actors字段里。

- “Calculate Actor weighting factor”步骤是计算器类型步骤，运行除法 $1/\text{count_actors}$ ，然后将结果保存到actor_weighting_factor字段。

刚刚我们得到了actor_key和actor_weighting_factor，但这些数据仍然还在它们自己的流中。

“Lookup actor weighting factor”步骤的任务可以把它们重新连接起来。这个步骤是“流查询”

类型的步骤，它使用从“Lookup dim_actor”步骤获得的film_key，从“Calculate Actor weighting factor”步骤中寻找匹配的行，从而将actor_weighting_factor字段从“Calculate Actor weighting factor”步骤传递到“Lookup actor weighting factor”步骤的输出流中。

执行完“Lookup actor weighting factor”步骤之后，我们有一个包含film_key、actor_key和actor_weighting_factor字段的数据流。使用一个简单的“插入/更新”步骤，这个数据流就可以加载到dim_film_actor_bridge表中。

load_fact_rental 转换

load_fact_rental转换是作业load_rentals最后的转换。作业load_rentals里所有之前的转换都是加载维度表，此转换是ETL最后的加载 fact_rental事实表。该转换如图4-18所示。

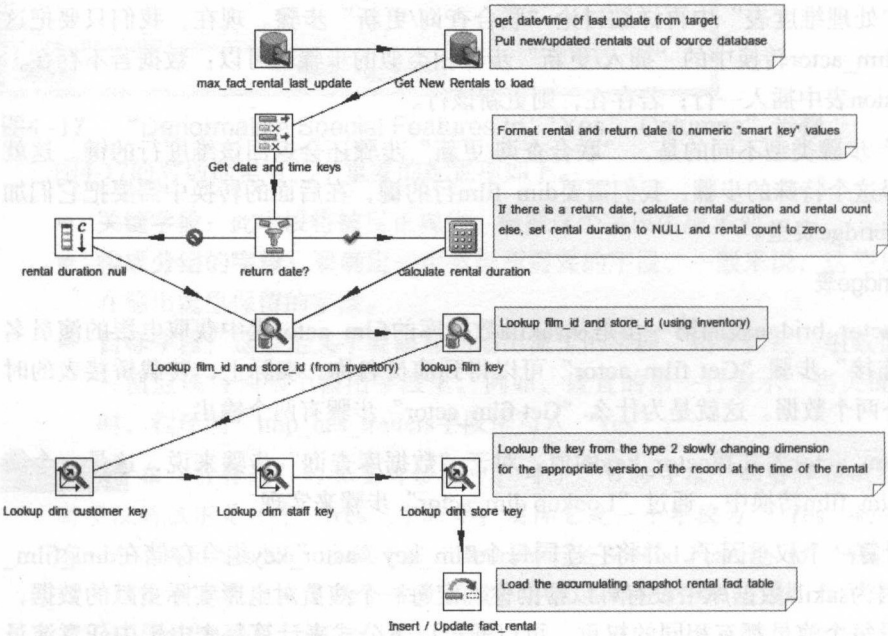


图4-18 load_fact_rental转换

load_fact_rental转换的结构与load_rentals作业中其他的转换不同。其他转换大多将数据反规范化，从而加载维度表，而此转换的本质在于计算度量值和查找相应的维度键。因此，尽管load_fact_rental转换几乎没有应用任何新的步骤类型，但仍然需要介绍一下它的关键设置。

变更数据捕获

转换的开始部分仍然使用两个“表输入”步骤。分别是max_fact_rental_last_update和步骤“Get New Rentals to load”步骤。

与其他转换相比，有个不同的地方，这个转换使用 rental_id 列来检测变化，而没有使用last_update列。因为 rental_id 值可以自动递增，所以 rental_id 列与 last_update 列的原理相同。唯一的缺点是：使用 rental_id 时，我们不能捕获到 rental 表的任何更新。这里，我们假设源表不存在更新的情况。

获取日期时间智能键

在我们讨论load_dim_date转换和load_dim_time 转换时，提到了dim_time和dim_date维度表

使用了智能键。智能键的值实际就是把日期和时间值转化成特定的字符串格式，通过步骤“Get date and time keys”来获得。这样在检索维度表的代理键时，不用再去查询维度表。

计算度量值

fact_rental表有3个度量值，如下所示：

- **count_rentals**：租借次数，这是所谓的非事实性事实。根据定义，如果在原rental表中有一行数据，就在 fact_rental 事实表中看作是一次租借。在数据库上，给该度量列设置默认值1即可实现。
- **count_returns**：归还次数，本列用来设置租借是否已归还，值是NULL（意味着迄今为止此租借没有归还）或1（意味着此租借已归还）。
- **rental_duration**：租借时长，本列用来设置租借时间长短，值的设置取决于租借是否已归还，值是NULL（意味着迄今为止此租借没有归还）或租出日期和归还日期之间的时长，单位是秒。

因为 fact_rental 表同时记录租赁和归还，随着时间变化某个数据行的状态度量值可能改变。这是一个简单而完整的累计快照事实表的例子。

过滤步骤“return date?”负责检查是否有归还日期。如果有归还日期，数据流向计算器步骤“calculate rental duration”，将count_returns度量值设置为1，再计算rental_duration的度量值，即用归还日期减去租借日期。如果没有归还日期，数据流向“rental duration null”步骤，该步骤将度量值count_returns和rental_duration设置为空。

查询常规维度表中的键

在原sakila数据库中，rental表没有直接指向store表或film表，而是有一列inventory_id，指向inventory表，再由inventory表指向store表或film表。所以，当我们查找维度的键值时，我们首先需要查找库存。通过“Lookup film_id and store_id (from inventory)”步骤查找库存。得到film_id和store_id后，就可以再查找真正的维度键。

查询类型2缓慢变化维度表的键

对于类型2的缓慢变化维度来说，不能简单地用原始的ID列查找维度，因为在类型2缓慢变化型维度中，每一个这样的ID值可以有多行。如果要从类型2缓慢变化维度表中查找行，需要知道ID以及上下文时间，这样才能找到正确的维度行。

如图4-18所示，上述功能是通过“Lookup dim customer key”、“Lookup dim store key”和“Lookup dim staff key”等步骤实现的，它们都是“维度查询/更新”类型的步骤。在load_dim_customer、load_dim_store和load_dim_staff这些加载维度的转换中已经遇到过这种步骤。但这一次，这种类型的步骤是用于在加载事实表时，从维度表中查找维度键，而不是用来向维度表中加载维度。在第8章中，将进一步说明该步骤。

加载事实表

转换的最后一步是“插入/更新”步骤。因为这是一个累计快照事实表，所以我们需要在租赁状态发生变化时（即当有租借或归还业务发生时）能更新事实表。

4.4 小结

这一章，我们介绍了很多内容。我们使用了一个相当简单的示例源数据库，解释了使用Kettle加载星型模型的关键技术。本章涵盖：

- 虚拟的Sakila DVD租赁业务的业务流程以及sakila示例数据库的结构和用途。
- Sakila租赁业务星型模型以及它与原始sakila示例数据库的关系。
- 基本的Spoon技巧，包括如何打开并运行作业和转换、如何设置数据库连接、如何编辑作业项和转换步骤，以及如何检查步骤/作业项之间连接。
- 用作业来组织转换的流程，成功和失败都发送电子邮件。
- 使用“表输入”步骤获取数据库里的数据，并通过SQL参数化设计一个简单的变化数据捕获方案。
- 使用值映射步骤转换值。
- 加载类型2的缓慢变化维度和查找维度键。
- 如何将转换作为子转换来调用以实现复用。
- 用“插入/更新”步骤加载类型1的缓慢变化型维度。
- 如何将一个分隔符分割的值字符串正规化为行，然后再把行转换成列。
- 计算聚集和度量。

本章讨论了上面这些内容，重点是在全局的流程上。下面的章节将讨论和这个例子相关的一些概念，以及Kettle更多的功能和特性。

第二部分：ETL

第2章

本部分包括

第5章 ETL子系统

第6章 数据抽取

第7章 清洗和校验

第8章 处理维度表

第9章 加载事实表

第10章 处理OLAP数据

第5章 ETL子系统

听起来也许令人难以置信，直到几年前市场上还没有专门介绍ETL系统的图书。ETL只是作为BI系统的一部分来介绍，但是很多人都希望能深入了解ETL的解决方案，不依赖于某个工具。Ralph Kimball和Joe Caserta编写的*The Data Warehouse ETL Toolkit*一书（Wiley 2004）填补了这一空白。随后，这本书里的一些思想形成了一篇文章“ETL里的38个子系统”，系统总结了ETL项目要面临的不同任务。

说明：我们还可以在网上找到原始的这篇文章 <http://intelligent-enterprise.informationweek.com/showArticle.jhtml?articleID=54200319>。这篇文章的最新版本在Kimball Group Reader, article 11.2 “The 34 Subsystems of ETL” 430~434页（Wiley 2010）。本书里子系统的名字都来源于后面这篇文章，和最初的名字相比有一些轻微变化。

在2008年，Wiley出版了最流行的一本BI图书的第二版：也是由Kimball和他同事编写的*The Data Warehouse Lifecycle Toolkit*。在这本书里ETL子系统被重构成了34种子系统。我们很幸运得到了Kimball的允许，将把这34种子系统作为本书第二部分的基础，在第二部分里，将向读者展示如何使用Kettle实现这34种子系统。本章将作为概要章节，在其他章节中再深入讲述如何实现特定子系统。本章帮助你理解如何把Kimball的理论基础转换为Kettle的具体实现。

本章假设你已经熟悉了维度模型和数据仓库的概念。如果你还不了解这些概念，参考我们第一本书*Pentaho Solutions*（Wiley, 2009）的第二部分。另外也强烈推荐前面提到的Ralph Kimball的经典之作*The Data Warehouse Toolkit, Second Edition*一书（Wiley, 2002）。

5.1 34种子系统介绍

正如*The Data Warehouse Lifecycle Toolkit*一书中描述的，这34种ETL子系统提供了一套框架，

帮助我们理解ETL解决方案的实现和管理，并对其进行分类。在开始ETL解决方案之前，我们需要清楚地了解需求、已存在的系统、可用的技巧和技术以知道我们的预期是什么，以及达到预期的推动和限制因素。这34个子系统中的很多都是管理类型的子系统，主要是因为当项目结果发布的时候，系统的生命期才刚刚开始。管理是子系统4个组成部分的一个，子系统的4个组成部分介绍如下。

- **抽取**：在第1章ETL介绍部分，我们已经介绍了ETL的数据抽取部分是ETL工作的一个最大的挑战，子系统1~3属于这个主题。
- **清洗和更正**：无论使用什么数据仓库架构，在某个时间点上，数据都要经过清洗以满足业务的要求。在Kimball的数据仓库里，在数据进入到数据仓库之前就被清洗了（“真实的数据只有一个版本”）。而如果使用Data Vault模型，数据是按照原样进入数据仓库的（“事实的数据只有一个版本”），而清洗和更正过程发生在后面的阶段。子系统4~8属于这个主题。
- **发布**：34个子系统中的13个都是关于如何把数据发布到目标数据库中的，发布并不仅仅意味着把数据写入到目标数据库中，也包括把数据写入到维度表或事实表中的那些转换。
- **管理**：正如前面提到的，企业的信息基础架构都要可以被管理和监控，ETL也不例外，子系统22~34属于这个主题。

5.1.1 抽取

ETL方案的第一部分就是要从不同的数据源抽取数据，正如在第1章中介绍的，访问数据源会有很多困难，政策性问题是难以逾越的障碍。另外基于安全性和性能方面的考虑，数据系统的管理人员不会让未经授权的用户访问系统。一些ERP系统的厂商（SAP或Oracle）也不允许其他系统访问ERP底层数据库，否则，作为惩罚，这些厂商甚至拒绝再提供服务。

子系统1~3：数据剖析，增量数据捕获和抽取

因为第6章会深入讲解这些主题，所以我们这里只简单介绍一下。

- **子系统1：数据剖析系统**——这个子系统的目标就是要分析不同数据源的结构和内容。数据剖析提供了类似行统计、NULL值个数统计等简单的统计项，当然也有一些更复杂的分析，如单词模式分析等。在编写本书时，Kettle只提供了有限的数据剖析功能，但可以使用Kettle作为其他此类开源项目的入口，通过其他开源项目来做此类统计工作，我们将在下一个章节中详细讲述。
- **子系统2：增量数据捕获系统**——这个子系统的目标是捕获源系统里数据的变化，当前Kettle里没有特别的工具可以完成这一功能，但Kettle里的一些步骤可以通过时间戳、快照的方式获得变化的数据。
- **子系统3：抽取系统**——抽取子系统就是要从不同的数据源抽取数据，并输入到ETL流程里。Kimball明确区分了基于文件的和基于流的两种抽取，注意这里基于流的抽取并不意味着实时数据流。从Kettle的角度看，这种区分方法不太恰当。因为无论从数据库、文件、实时数据源、Web Services还是其他任何数据源，只要可以访问到数据，数据都是以流的方式通过整个转换。事实上唯一有区别的地方是在Kettle运行作业的过程中，数据源的数据是否在发生变化。所以抽取的主要的区别不是文件或流，而是静态或动态的问题。如果转换失败，这种区分方式就显得更为重要。在第1章解释过，任何

ETL解决方案都要考虑到失败的情况。如果你的数据源是静态的（文件的情况基本都是如此），重新启动一个作业就可以了。而如果你的数据源是动态的，例如事务型的数据库，在你运行作业的时候，数据库里面的数据已经发生了变化。例如，一个加载销售数据的作业，所有的维度都正常加载，在加载销售订单的事实数据时，停电了。而在加载维度数据的同时，源系统中有了一个新的客户，并产生了一个新的订单，也就是说在源系统中有了新的维度数据和事实数据。除非把所有的维度表重新跑一遍，否则在重新加载事实表时，就会发现有的客户维度没有找到。另外，CDC的实现方式不同，从这类错误中进行数据恢复也是非常困难的事情。

5.1.2 清洗和更正数据

世界上没有任何一个组织的数据是没有质量问题的，这也就是为什么我在把数据加载到数据仓库之前要增加一些步骤来清洗和更正这些数据，以满足业务的需求。另外，只使用一个单一的系统来存储数据的组织也很少；通常，为了支撑业务运行，都会存在多个系统，可能每个部门都会有自己的系统，如财务、人力、采购或客服管理等。每个系统存储数据的方式可能都不相同。例如在系统A里，客户性别保存为F（female）、M（male）、U（unknown）；在系统B里，分别使用0、1、NULL来代表这三类数据。所有的这些系统都应该遵照数据仓库的统一标准。

子系统4：数据清洗和质量处理系统

数据清洗是指修改或整理进入到ETL流程中的脏数据。尽管我们反复强调数据清洗应该在原始系统中产生数据的地方进行。但往往提高原始数据质量所需要的时间不能满足开发数据仓库的时间要求。但是无论如何，我们都要给用户一份干净的数据。所以一般就需要使用ETL项目来提高数据质量，ETL项目的优势在于：首先，在ETL的数据剖析阶段，可以找出有哪些错误数据；其次，在源系统中需要的数据清洗规则，同样可以使用于ETL环境中。最后，最终使用数据的业务人员可以加入到ETL开发中，只有业务人员才能告诉我们哪些数据是正确的数据。

理想情况下，业务人员/数据所有者、源系统开发人员/管理者和ETL开发人员需要共同完成提高数据质量的工作。在很多情况下，不正确数据主要来源于那些把数据输入到系统里的业务人员。

例如我们的一个客户，我们最近给他们开发了一套包括ETL流程的数据质量解决方案，这个方案读取并转换业务系统中的数据，最后把数据加载到一个检查系统，在这个检查系统里用户可以可视化查看数据，并给不正确数据打标记。另外，ETL流程还可以自动给某些常见的错误打标记，如字段为空、不正确的格式或错误的电话号码等。每周数据质量的检查结果就会报告给数据管理人员。尽管业务上要求100%没有错误数据，但实际上，在没有做这个数据质量项目之前，正确数据的比例低于50%，在做了这个可视化的数据质量项目后，第一年，正确数据的比例已经几乎达到了90%。这个例子显示了一个简单的ETL流程再加上一些报告，如何让用户重视并提高数据质量问题。

在第7章，可以看到如何使用 Kettle实现上面案例的效果，以及数据清洗和确认的一些场景。

子系统5：错误事件处理

第7章还介绍了错误事件处理。错误事件处理的目的是记录下ETL过程中的每一个错误。这

样便于管理员定期监控和分析错误，是数据质量错误，还是系统错误或其他错误。Kimball 提到要使用一个独立的错误事件模式来保留这些错误，但是在Kettle 里，并不需要这样的模式，在第7章我们就可以看到，Kettle有很多现成的功能来处理错误事件日志。

子系统6：审计维度

尽管错误事件模式和数据仓库的业务数据是独立的，但审计维度表却是数据仓库内部的维度表。审计维度表是一类特殊的维度表，数据仓库里的所有事实表都和审计维度表关联，审计维度表包含了对事实表变更的元数据，如加载数据的日期和时间、数据的质量指标等。实际上，给数据仓库增加审计维度，可以带来很多好处。就像在多维数据仓库上使用 Data Vault（第19章）架构所带来的好处一样。审计维度将在第7章介绍。

子系统7：排除重复记录系统

排除重复记录可能是ETL项目中最棘手的问题，大部分ETL工具也没有能自动处理重复数据的功能。在大多数情况下，排重是指删除重复的数据，或者把不同系统里互相冲突的数据统一。客户数据是最容易产生重复数据的地方，其他类型的数据也可能会有这样的问题。只要是非常大量的参照类型的数据，就都有可能存在重复数据，例如产品或供应数据。

如果你熟悉微软的Access，你可能会觉得排重是一件很简单的事情，只要使用“查找重复记录的查询向导”就可以解决。实际上，如果只是在关键字上完全相同的重复记录，如完全一样的电话号码、街道地址等，这类重复问题的确非常容易解决。但事实并非如此简单，名字、地址可能都会拼写错误，或使用了缩写，电话号码也可能写错，一个系统里更改了电话号码而另一个系统里没有更改，等等。在大多数情况下，可以采用模糊查询、正则表达式匹配、Soundex函数，以及各种数据挖掘技术来解决这类问题。要想判断出“JCJM VanDongen”和“Dongen, van Jos”是一个人，或者“Bouman RP”和“Roland Bouman”是一个人都是很艰巨的任务。有几个高级的（而且昂贵的）工具可以把这类事情做得比较好，遗憾的是Kettle并不是这些工具之一。但第7章还是介绍了如何使用Kettle处理重复数据。当然也可以使用类似DQ Guru这样的工具，这个工具由 SQLPower 公司开发，它是一款专业的工具，帮助你去除重复数据。DQ Guru也是开源的，关于它的更多介绍参考：<http://www.sqlpower.ca/page/dqguru>。

子系统8：数据一致性

数据经过数据排重子系统和前面提到的其他数据质量步骤处理后，就交给数据一致性子系统来处理。这个步骤的目的就是使来源于多个业务系统的事实数据遵照相同的维度。例如，一个公司有一个客服管理系统，这个系统有自己的客户数据库，为了把客服管理系统和销售系统放到同一个数据仓库里，需要把客服管理系统的客户数据和销售系统的客户数据统一成一个客户维度表，当分别加载来自这两个系统的事实数据时，需要把来自两个系统的事实数据指向同一个客户维度表。解决这个问题最常用的方法就是维度表中保留从不同系统带来的自然键，在加载事实数据时，可以查找维度表中的这些源系统中的自然键，第9章详细讲解了在Kettle里如何查找和加载事实数据。

5.1.3 数据发布

发布新的数据并不只是往目标数据库里插入新的数据这么简单，发布新的数据其实有很多工作。首先，我们从不同缓慢变更维度技术可以看到，更新维度表就有很多种方式。另外，你需要生成代理键、查询正确的维度键、确保维度数据在事实数据加载前就已经加载完、准备要加载的事实数据。加载事实数据本身也是一项有挑战性的工作：事实数据可能数据量比较大，也有可能还要更新事实数据，或者同时出现这两种情况。所以需要特别关心表和存储的功能，如OLAP数据库。这也是为什么34种子系统里有很多都属于数据发布范畴。

子系统9：缓慢变更维度处理

缓慢变更维度（SCDs）是多维数据仓库或者总线架构的基础。我们知道维度表里保存了用来对事实进行分析或分组的信息。例如，客户维度里有客户所在城市字段，这样我们就可以统计或列出某个城市里客户的销售情况。如果客户换了另一个城市，业务系统里肯定要相应修改这个客户所在的城市，缓慢变更维度的过程也会根据不同的规则来变更数据仓库中的客户维度。总的来说，有以下3种缓慢变更维度的方法。

- 覆盖：直接用新值代替旧值。
- 增加新行：把当前行标记为“old”并设置一个“结束”时间戳，同时创建一个新行，标记为“current”，并设置一个“开始”时间戳。
- 增加新列：给表增加一个新列，来存储新值，同时保留原来的值不变。

在Pentaho Solutions一书的第7章，我们又在上面的基础上扩展了如下的另外3种方法。

- 增加一个小维度表：把经常变更的表属性从主维度表里分离出来，保存在自己的表里。
- 分离历史表：把每次变化保存到一个历史表里，同时保存变化的类型和变化的时间。这样的历史表可以回答类似“去年有多少客户从佛罗里达移动到了加利福尼亚”这样的问题。
- 混合型：把类型1、2、3结合起来（1+2+3=6）。

第8章将介绍前3种缓慢变更维度的方法，并说明如何使用 Kettle来实现这些方法。

子系统10：代理键生成系统

ETL流程应该可以生成代理键。使用Kettle可以非常容易生成代理键，Kettle里的“增加序列”步骤可以自动生成或使用数据库里的序列来生成代理键。这种步骤适用于创建或第一次加载维度表，而不适用于以后修改维度表。可以使用“维度查询/更新”步骤和“联合查询”步骤来解决更新维度的问题。这两个步骤都有3种方式来生成代理键，如下所示。

- 使用表里现在代理键的最大值+1。
- 使用数据库序列。
- 使用一个自增的字段。

后面一种方法也可用于表输出步骤。

子系统11：层次维度构建

在数据仓库里还要考虑如何构建和维护数据仓库里的层次。实际上，这个子系统的完整的

名称是“构建固定的，可变深度的，可有级别缺失的层次维度系统”。层次可以让用户分析查看维度不同级别上的数据。最简单的层次概念就是时间维度的层次。在现实中，时间维度都需要至少一个以上的层次。例如有“年—季—月—日”这样的层次，也有“年—周一日”这样的层次。时间维度也是“平衡层次”的一个例子。在时间维度里，所有级别的深度都是固定一样的。组织结构的维度更复杂一些，这种维度通常都是“不平衡的”或称为“可变深度的”（子树的深度不同）或“级别缺失的”（在层次上缺失了一些级别）。关于后面“缺失的”，可以想一下地理维度，地理维度通常的层次是“国家—地区—州（省）—城市”。一些国家可能没有“地区”或“州（省）”或都没有，这就是缺失了级别。在源系统里，通常使用“递归”的关系来实现“不平衡的”或“级别缺失的”的情况。Kettle 提供了一些（不是全部）功能，可以把这类层次结构扁平化，在第8章可以看到。

子系统12：特殊维度生成系统

除了缓慢变化维度，基于多维模型的数据仓库，至少都包含一个特殊维度，时间维度。下面一些类型的维度也都是特殊维度。

- **杂项维度（也称为垃圾维度）**：一些零散的属性，分析需要但又不适合放在其他维度表里。例如状态标志、yes/no和其他低阶（low cardinality）字段都可以放在杂项维度表里。
- **小维度**：从大维度表里分离出经常发生变化的一些属性，单独放在一个小维度表里。我们也把这种小维度表称为 SCD 4。
- **收缩的或上卷的维度**：普通维度表的子集，为了避免冲突，这种维度是根据普通维度表创建和更新的。这种维度适用于聚集的数据，例如底层保存的是每天的数据，聚集的数据是按月保存的。
- **静态维度**：通常是小的字典表或参照表，这类表在源系统中没有对应的数据，如状态编码描述或性别。
- **用户自定义维度**：源系统里没有的而报表需要的自定义的描述、分组和层次。可以是任意的维度。唯一区分它们的就是这些维度是通过用户来维护的，而不是通过数据库团队或一个ETL过程。

第8章将详细讲述维度的加载方法。用户自定义维度在处理上不同，一般也需要一个应用程序来维护这些用户自定义的维度，这超出了本书的范围。但提个建议，可以看看Wavemaker这个开源项目，这是一个快速应用开发工具，它可以在几分钟之内开发一个维护界面，项目网址：<http://dev.wavemaker.com>。

子系统13：事实表加载

在往数据仓库加载事实表之前，需要把数据准备好。加载事实表过程并不是重点，之所以把加载事实表单独作为一个子系统分出来，主要是为了强调如下三种不同类型的事实表。

- **事务粒度事实表**：以每一个事务或事件为单位，例如一个销售记录、一个电话呼叫记录，作为事实表里的一行数据。
- **周期快照事实表**：事实表里并不保存全部数据，只保存固定时间间隔的数据，例如每天或每月的库存水平，或每月的账户余额。
- **累积快照事实表**：当有新的数据时，更新事实表里的记录。数据仓库里总是保存最新的

数据。例如订单过程，订单过程里有很多独立的日期，如订单日期、期望发货日期、实际发货日期、期望收货日期、实际收货日期和付款日期。当这个过程进行时，随着上面各种时间的出现，事实表里的记录也在不断更新。

（请注意在34个子系统这个文档的最新版本中，这个子系统的全名是“事务粒度、周期快照粒度、累积快照粒度事实表的加载系统”。）加载事实表，通常要加载几百万行数据。为了快速加载，大多数数据库系统都提供了批量加载方式，批量加载方式通常规避了数据库的事务引擎，直接把数据写入到目标表。有时为了提高处理数据的速度，要删除事实表上的所有索引，在加载完后再重建索引。第9章深入介绍了不同的事实表加载技术。

子系统14: 代理键管道

这个子系统负责抽取正确的代理键，用于加载事实表。这里用“管道”一词是因为事实表的加载看起来像一个工序，工序里的每个环节都使用数据的自然键去查找维度表里的代理键。为了让这个查询过程更高效，最好把要查询的维度数据预先装载到内存里。第9章介绍如何使用Kettle的“流查询”步骤和“数据库查询”步骤来查询代理键。

子系统15: 多值维度桥接表生成系统

处理不同深度的层次时需要桥接表，例如一个客户，是一个公司，它有子公司和子子公司。每一级的公司都可能去购买商品，如果想从母公司的角度去看一共购买了多少商品，就需要使用桥接表来实现。当有多个维度项和事实表或其他维度表关联时，也要使用桥接表。例如：电影票和电影演员，如果想汇总一个演员有多少电影票收入，就需要在电影和电影演员维度之间建立一个桥接表，这个桥接表把电影和电影演员关联起来，桥接表里还可以设置电影演员的权重因子。对于构造和维护桥接表，Kettle 没有提供什么特殊的功能，但在第4章的sakila转换里我们演示了一个例子。

子系统16: 迟到数据处理

到目前为止，我们的讨论都是在要处理数据同时到达的假设前提下。但在一些场合下，并非如此：事实表数据和维度表数据都可能晚到。对事实表来说这不是什么大问题，唯一不同的就是要根据维度的有效时间查找业务发生时的维度代理键。只要在查询条件里增加 valid_from 和 valid_to 两个字段就可以。“维度查询和更新”步骤默认就有这两个字段。如果维度表数据晚到，情况就要麻烦一些。如果事实表已经加载完了，但维度表的数据不是最新的。当要更新的维度数据过来后，按照 SCD 2，会在维度表里增加一条记录，此时要使用新创建的维度的代理键来更新事实表里有上一个代理键的数据。另外还有一个方法，当事实数据过来，但根据事实表里的维度自然键，从维度表里找不到对应的代理键。此时先创建一个新的维度记录，所有的字段都设置成默认值和空值，使用这条记录的代理键。然后当正确的维度数据从源系统中过来时，再更新这些默认值和空值。迟到维度数据在第8章介绍，迟到事实数据在第9章介绍。

子系统17: 维度管理系统

在 34 个子系统的文章中是这样描述这个子系统的：“中心控制系统，用来准备和向数据仓库发布正确的维度”。中心控制系统不只是组织，还负责管理所有和维度相关的任务。在第8

章，将介绍如何使用Kettle 组织和管理维度表。

子系统18: 事实表管理系统

这个子系统负责任何创建、组织、管理和事实表相关的任务。子系统17和18在一起结伴工作: 事实表管理系统获取到由维度管理系统管理的维度, 并把这些维度放到事实表中。第9章介绍关于这个子系统的更多内容。

子系统19: 聚集构建

如果数据库是用于分析的, 一定会有性能方面的要求。这种对速度的要求产生了几种解决方案, 在这几种方案里, 聚集表对性能的提升最大。如果能把平均30分钟的响应时间降低到几毫秒, 客户会非常高兴。聚集表就可以达到这样的效果。但仅有聚集表是不行的, 还需要维护聚集表 (Kettle也可以做这样的事情), 数据库还需要知道聚集表的存在以利用聚集表。这也就是MySQL、PostgreSQL、Ingres这些开源产品和Oracle, SQL Server及DB2这些商业产品的差距所在 (这些商业产品都有自动聚集导航功能)。有聚集表功能的唯一的一个开源产品是Mondrian, 但这些聚集表还是需要由Mondrian聚集表设计器来创建和维护。另外, 也可以使用特殊的分析型数据库, 如LucidDB、InfoBright、MonetDB、InfiniDB、Ingres/Vectorwise, 或把分析型数据库如LucidDB和Pentaho 聚集表设计器结合起来。生成和加载聚集表数据只是一次性的工作, 但当数据仓库的数据发生变化后, LucidDB和Pentaho聚集表设计器都不会去维护聚集表。在第9章我们介绍如何使用Kettle维护聚集表。

子系统20: OLAP Cube构建系统

OLAP数据库有特殊的存储结构, 当加载的时候, 可以预先聚集数据。一些OLAP数据库只能写不能更新, 所以, 在做更新之前要把源数据清除。其他OLAP数据库 (如微软的分析服务器) 可以更新事实表, 但有它自己的更新机制, Kettle不能使用。第10章主要介绍如何处理OLAP数据以及如何使用Palo插件向Palo cube 里加载数据。

子系统21: 数据整合管理系统

这个子系统用来从数据仓库获取数据, 并把数据发送到其他环境中, 通常用于离线数据分析或者其他特殊目的, 如给特定客户发送报表。在第22章, 我们会介绍如何使用Kettle来实现这一功能。

5.1.4 管理ETL环境

本章的最后将介绍14个ETL子系统, 这些子系统用来完成管理功能。本书的第三部分将更深入地介绍这些子系统, 这里只对每个子系统做些简要的介绍。

- 子系统22: 作业调度——社区版的Kettle 不提供自己的调度功能, 而是依赖于Pentaho BI 的调度功能或者操作系统的cron功能。第12章将详细介绍调度和作业日志。
- 子系统23: 备份系统——备份ETL过程中产生的中间数据也应该是ETL方案的一部分。Ralph Kimball 推荐在ETL流程中的三个地方缓存 (备份) 这些数据: 1) 从源系统中抽

取之后, 做任何改动之前。2) 清洗、排重、更正之后, 此时可能还在文本文件中或使用正规化的格式。3) 已经做完最后处理, 可以写入到数据仓库之前。备份数据仓库本身通常不是ETL团队的工作, 但ETL团队可以和DBA紧密合作来实现错误恢复的方案。

- **子系统24: 恢复和重新启动系统**——ETL设计的一个重要部分就是在ETL失败时, 可以重新启动。我们要尽量避免丢失数据和重复数据的情况, 所以这个子系统非常重要。遵照前面子系统描述的策略可以更容易重新启动一个失败的作业。在第11章里, 作为测试和调试主题的一部分, 我们再详细介绍这部分内容。

- **子系统25: 版本控制系统; 子系统26: 从开发环境到测试、生产环境的版本移植系统**——有很多种方法可以实现版本控制。第13章更深入地介绍这一主题。Kettle企业版有内置的版本控制系统。对于Kettle 企业版和社区版都可以使用SVN或CVS这样的版本控制系统。版本控制系统也不应该成为一个事后才想到的问题。在这里引用Ralph Kimball对版本控制系统的观点。

你需要给ETL系统里的每个部分都确定一个主版本号, 另外ETL系统作为一个整体也要有一个主版本号。这样如果今天发布的版本发生严重的错误, 可以快速恢复到昨天的ETL版本。

The Data Warehouse Lifecycle Toolkit, 2nd Edition,

By Ralph Kimball et al., Wiley, 2008

- **子系统27: workflow 监控**——是否尝试过不使用计时器和温度指示器来烤蛋糕? 非常难吧? 同样运行一个ETL作业, 而不使用任何方法去监控运行过程, 不显示执行作业的执行细节, 也会使运行ETL作业非常困难。已经处理了多少行, 处理的速度有多快? 消耗了多少内存? 哪条记录出错了, 为什么出错? 监控过程应该可以回答所有的这些问题, 在Kettle里, 日志框架就是监控过程, 第12~14章将详细描述Kettle的日志框架。
- **子系统28: 排序系统**——对于一些操作(如分组、排序合并操作), 数据事先要进行排序。当然, Kettle 有一个“排序”步骤, 这个步骤在内存里操作, 但如果数据太大了会在硬盘上分页。对于非常大的文件, 可以需要独立的排序工具。我们不讨论这些专用的排序工具而只是使用我们的“排序”步骤来完成我们的排序工作。
- **子系统29: 血统和依赖分析**——ETL系统应该同时提供血统分析和影响分析功能。血统分析从处理后的数据开始向后追溯查看这个数据起源于哪里, 然后在中间的环节对这个数据进行了哪些处理。依赖或影响分析的方向和血统分析相反, 即从数据的起源开始, 查看哪些步骤或转换使用了这个数据, 这样可以显示出如果一个数据或表发生了变化会影响到系统的哪些部分。在编写本书时, Kettle有显示出步骤对数据库影响的功能, 但是完整的血统和依赖性分析还要在Kettle的未来的企业版本中发布。第14章介绍了使用当前版本的Kettle 元数据都能做什么事情。
- **子系统30: 问题报告系统**——万一运行中出了错误(相信我们, 肯定会出错的), 你需要尽快知道运行中发生了错误。第7章介绍了错误处理和通知。
- **子系统31: 并行/管道系统**——为了能在短时间内处理大量数据, 任务应该可以并行运行, 甚至在多台机器上同时运行。第15章和16章介绍了这一主题。这里特别要提到Kettle 易于使用的集群功能, Kettle集群可以动态扩展(如晚上ETL任务工作的时候), 也可以在不需要的时候关闭(如作业运行完)。在云计算环境下(如Amazon的EC2)运行ETL作业, 可以避免大规模的硬件投资, 以很低的运营成本带来大规模的按需可扩展的计算能力。

- **子系统32：安全系统**——在现在的IT领域，安全和合规是很热门的主题。数据仓库里保存了企业所有的数据，也是最容易产生风险的地方。另外在很多情况下，ETL过程可以直接访问到很多源系统，所以ETL解决方案本身也是易被攻击的地方。
- **子系统33：合规报告系统**——确保一个 ETL流程遵照规章制度，所需要的大多数方法已经在其他子系统中涉及到了。合规意味着要对数据进行详细的审计，审计包括数据从哪里来，在数据上面执行了什么操作（血统），数据在写入到数据仓库之前是什么样子（基于时间戳的备份），在每个时间点的值是什么（审计表，SCD 2），谁访问了数据（日志）。Data Vault是一种提供了很好审计功能的数据模型，在第19章介绍。
- **子系统34：元数据资源库管理系统**——这个子系统的目标就是捕获到和ETL相关的所有业务、过程和技术元数据。这个子系统中重要的一部分就是把系统文档化，在第11章介绍，当然Kettle的全部架构也是元数据驱动的，我们曾经在第2章讨论过。

5.2 小结

本章介绍和解释了Ralph定义的34种ETL子系统，并把这些子系统联系到Kettle的组件和本书相关的章节。这些子系统的列表也可看成是ETL架构的通用的定义：它描述了每个子系统应该去做那些工作，而不是如何去做或者拿什么工具去做。因此不止是验证Kettle，它实际上是验证所有ETL解决方案的一个详细需求列表。这34种子系统涉及的四个主要方面如下。

- **抽取**：从不同的数据源里获取数据。
- **清洗和更正数据**：转换和集成数据，为数据进数据仓库之前做准备。
- **发布数据**：加载和更新数据仓库里的数据。
- **管理环境**：控制和监控ETL解决方案所有组件的处理过程。

正如本章所介绍的，有一些功能 Kettle 还不能全部支持（主要是血统分析和影响分析），但总的来说，Kettle是一款优秀的数据整合工具软件。

第6章 数据抽取

ETL过程的第一步就是从一个或多个数据源获取数据，在第1章和第5章曾介绍过，数据抽取是一个艰难的工作，因为数据源是多样和复杂的。在传统的数据库环境下，数据通常来源于企业的事务类应用系统，如财务系统或ERP系统。大部分这类系统都是把数据存储在关系数据库中，如MySQL、Oracle或SQL Server。抽取一般要从业务的角度来抽取，这也是一个挑战（在本章的后面我们看一个ERP系统的例子），从技术上来看，最好能使用JDBC直接连接数据库。但如果数据库不是关系型的或者没有可用的驱动，数据抽取就会更有挑战性。在这种情况下，一般就需要使用逗号分隔的文本文件来获取数据。还有一种情况就是数据属于其他人，可能是某个供应商或客户的数据，数据位于公司防火墙之外。在这种情况下，不可能直连，使用文本文件交换数据是唯一选择。如果数据位于互联网上，文本文件也不能使用。在本章的后面，可以看到Kettle提供了几种方法来访问互联网数据，如通过RSS或者Salesforce.com网站直连，或者通过Web Services等。

本章的第一部分讲述了Kettle中几种抽取数据的组件。后面的部分讲述数据特征，在ETL项目里非常重要的但容易忽视的部分。数据特征描述了数据的基本结构，它是数据的一组统计信息，让你从总体上了解数据内容和数据质量。Kettle本身也包含了一些获取数据特征的功能，但在本章介绍一个用来获取数据特征的专用工具——eObjects.org的DataCleaner。

本章的第三部分讲述 Change Data Capture (CDC)，以及 Kettle 如何支持不同的CDC 技术。本章的最后一部分讲述将数据传送到ETL流程的下个环节需要用到的一些技术。

6.1 Kettle数据抽取概览

Kettle 新手在第一次使用Spoon 时都会发现所有数据抽取类的步骤都放在“Input（输入）”

类别下，输入类的步骤，顾名思义就是从外部数据源抽取数据，把数据输入到Kettle的数据流中。另一件使大多数人都惊奇的事情是“输入”类别下有如此多的输入步骤。尽管这些步骤覆盖了Kettle的大部分数据抽取功能，但也并不是所有的功能。一般来说准备要读取的数据（尤其是文件类数据）的功能，往往在作业里完成，实际读取数据才在转换这一层。下面按照不同类别分别说明处理数据和数据抽取的功能选项。这里的说明是概要性的说明，详细说明参考使用Kettle在线帮助文档。在菜单条上选择“帮助”→“显示欢迎窗口”来打开在线帮助文档。

6.1.1 文件抽取

当在ETL过程里使用文件时，方法非常简单：在转换里提供了文件基本的读写操作，对于文件的其他操作（移动、复制、创建、删除、比较、压缩、解压缩）都在文件管理作业项中。

说明：在使用文本文件输出步骤前，不必先创建一个文件。如果文件不存在，文本文件输出步骤会自动创建一个文件。

处理文本文件

文本文件可能是使用ETL工具处理的最简单的一种数据源。读写文本文件没有太多技巧。文本文件易于交换，压缩比较高，任何文本编辑器都可用于打开文本文件（当然，除非文件有几个G这么大，那样只能用记事本打开）。总体来说有以下两类文本文件。

- **分隔符文件：**这种文件里，每个字段或列都由特定字符或制表符分隔。通常这类文件也称为CSV（逗号分割文件）文件或制表符分隔文件。
- **固定宽度文件：**每个字段或列都有指定的宽度或长度。尽管固定宽度文件的格式非常明确。但也需要一些时间来定义。Kettle也在固定宽度文件输入的“获取字段”选项里提供了一些辅助工具，但如果要在分割符文件和固定宽度文件之间选择，最好还是选择分割符文件。

对于这两种文件，都可以选择文件编码。UTF-8是通常情况下的标准编码格式，但其他编码格式，如US-ASCII或Windows-1252也在广泛使用。为了正确读取文件里的内容，必须要正确设置文件编码。不幸的是，在很多情况下，我们并不知道文件的编码，所以有必要规定文件的发送方和接收方都使用同样的编码标准。

提示：查看文件编码也有一些小窍门，如使用Firefox打开文件，选择“视图”→“字符编码”，或者打开IE，选择“视图”→“编码”，都可以查看文件编码。

最基本的文本文件输入步骤就是“CSV文件输入”步骤。CSV文件是一种用分隔符分割的文本文件。在处理这种文件之前，要确定分隔符和字段。如果不确定分隔符或封闭符，要先用文件编辑器查看该文件，判断分隔符。“CSV文件输入”步骤和与之类似的“固定宽度文件输入”步骤都不太适合一次处理多个文件。这两个步骤其实都是“Text file input（文本文件输入）”步骤的简化版本，“文本文件输入”步骤是一个功能非常强大的步骤，也是处理文本文件的首选步骤。“文本文件输入”步骤的主要功能如下。

- 从前一个步骤读取文件名。
- 一次运行读取多个文件名。
- 从.zip或.gz压缩文件中读取文件。

- 不用指定文件结构就可以显示文件内容。注意需要指定文件格式（DOS、UNIX、Mixed），因为Kettle 需要知道文件的换行符。
- 指定逃逸字符。用来读取字段数据里包含着分隔符的字段，通常的逃逸字符是反斜线（\）。
- 错误处理。
- 过滤。
- 指定本地化的日期格式。

当然用这些功能是有代价的，“文本文件输入”步骤比前面两个步骤要占用更多内存和CPU处理能力。

处理多个文件的例子

在这个案例里，我们看Kettle如何处理一个常见的场景。这个场景如下：

- 从参数表中获取目录名。
- 根据搜索字符串获取一组文件名。
- 读取这些文件。

我们用的例子文件是custfile1.txt和custfile2.txt，每个文件都有25 000行客户数据。从本书网站上可以下载这些文件www.wiley.com/go/kettlesolutions或者从仿造名生成器网站下载这些文件<http://www.fakenamegenerator.com>，下载后把它们放在不同的目录下。可以使用任意的目录名和文件名，但文件名需要以cust开头。

创建一个新的转换，在转换中添加一个“文本文件输入”步骤。首先要确定文件的结构，打开“文本文件输入”步骤的设置对话框，找到其中的一个文件，并把这个文件添加到选中的文件对话框中。在“内容”标签里设置正确的文件格式（DOS、UNIX、Mixed）。现在可以使用“显示文件内容”按钮打开这个文件，可以看到，这个文件使用管道符（|）作为列分隔符，有一行表头。我们就可以使用这些信息来设置“内容”标签里的“表头行数”、分隔符、封闭符等参数。

定义完文件格式后，再选择“字段”标签并单击“获取字段”按钮。Kettle会尽量判断出每个字段的数据类型。尽管一般情况下，Kettle 都会判断得比较准确，但也并非万无一失。以数字开头的字段（如类似 '2342 Wilwood Street' 这样的地址字段或类似 '23;44;33' 这样的字段）会被识别成整型，是不对的。在本例中，StreetAddress字段被认为是长度为4的Integer类型，需要改成长度为35的String类型。在本章的后面，可以看到如何使用数据特征（Data Profiling）功能来判断数据类型。

为了验证设置的正确性，单击“预览”按钮，如果出现预览的数据，说明设置正确，就可以关闭这个步骤了。现在我们要继续创建产生文件名的步骤，产生文件名的步骤要作为刚创建的“文本文件输入”步骤的前驱步骤。

首先，添加一个“获取文件名”步骤。注意在“选中的文件列表”里可以添加多个文件或目录。这里只需要输入目录名和匹配文件的通配符 ^cust.*，这个通配符将搜索所有以cust 开头的文件名。图6-1显示了该配置窗口。

现在可以预览一下，看是否获取到了正确的文件名，预览窗口如图6-2所示。

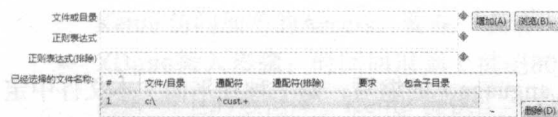


图6-1 获取customer文件

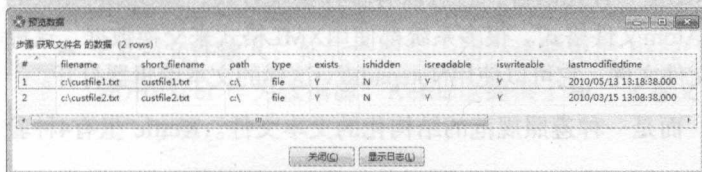


图6-2 预览获取到的文件名

下一步需要把“获取文件名”步骤和“文本文件输入”步骤连接起来。然后打开“文本文件输入”步骤，选中“从上一个步骤接收文件名”选项，并选中“获取文件名”步骤作为文件名来源步骤。选中 filename 字段作为文件名字段。注意现在不能再使用“预览”选项了，只能在该步骤上选择转换里的预览。我们注意到在“文本文件输入步骤”里也有路径和文件名正则表达式的选项，但最好把选文件的过程单独放在“获取文件名”步骤里。因为“获取文件名”步骤可以从前面的步骤获得路径名和文件名的正则表达式，如果路径名和文件名正则表达式存在数据库表中，这样使用“获取文件名”步骤来获取文件名就比较灵活了。为方便起见，我们使用“自定义常量数据”步骤来模拟数据库表输入步骤，在“自定义常量数据”步骤里创建两个字符串类型的字段（source_dir 和 source_pattern），并给这两个字段设置签名的路径名和文件名正则表达式字符串。设置好的“自定义常量数据”步骤如图6-3所示。

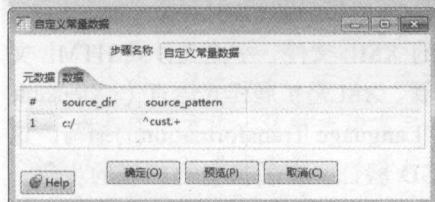


图6-3 设置好路径名和文件名正则表达式的“Data Grid(自定义常量数据)”步骤

此时，在“获取文件名”步骤里选中“文件名定义在字段里”选项，并选中“source_dir”和“source_pattern”字段，如图6-4所示。

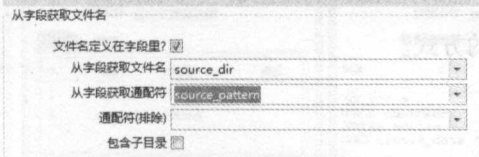


图6-4 “Get file names (获取文件名)”步骤

在生产环境中，应该使用“表输入”步骤来代替“自定义常量数据”步骤。另外还有一种方法来动态获取目录名和文件名，这种方法需要使用变量，要把现在的转换分成两个转换，第一个转换从数据库参数表里获得路径和文件名参数，并把参数赋值给变量，在下一个转换的“文本文件输入”步骤里再使用这些变量。

从这些例子里可以看出，可以用很多方法来做一件事情。这取决于解决方案的灵活性。一般要花更多的时间来设计和构建一个更灵活的参数化的解决方案，一个灵活的解决方案可以在开发、测试和生产环境中都能流畅运行。关于解决方案设计的更多内容可参考第11章。

处理XML文件

XML是扩展标识语言（eXtensible Markup Language）的缩写，是一种在平面文本文件中定义数据结构和内容的开放标准。尽管以 XML 为扩展名的文件可以说明这个文件是XML文件，但这只是XML的一部分。实际上 XML是一种元语言，它可以有多种实现方式，如OpenOffice的OpenDocument文件格式，或RSS、Atom文件格式。很多系统都使用XML格式来交换数据，这种XML格式非常流行。XML实际就是文件文件，它可以使用Notepad或vi这样的文本编辑器打开。

XML文件不是普通的文本文件，而是一种遵照规范的结构化的文本文件。Kettle 里有4种验证XML数据是否有效的方法。

- **验证XML文件是否有效**：基本的验证，只验证 XML是否有完整的开始和结束标签，各层嵌套的结构是否完整。
- **DTD 验证**：检查XML文件的结构是否符合DTD（Data Type Definition）文件的要求。DTD文件可以是一个独立的文件，也可以包含在XML文件中。
- **XSD 验证（作业）**：检查XML文件的结构是否符合XML Schema 定义文件的要求。
- **XSD 验证（转换）**：和上面相同，但XML是在数据流的字段里（如数据库的列里包含XML格式数据）。

检查完XML格式的正确性后，可以使用“XML输入”步骤读取XML文件。读取XML文件的主要障碍就是分析嵌套的文件结构。从这个步骤输出的数据流就是平面的没有嵌套的数据结构，可以存储在关系数据库中。与之相反，“增加XML列”字段用来把平面数据构造成嵌套形式的XML格式数据。

如果想把 XML 转成其他的任何格式——如另一种格式的 XML 文件、平面文件或 HTML 文件——这时就要使用“XSL transformation（XSL转换）”步骤。XSL是扩展样式语言（eXtensible Stylesheet Language）的缩写，XSLT是eXtensible Stylesheet Language Transformation的缩写，这是一种用来转换 XML 文档的 XML 语言。如果想使用 XSD 验证，在转换里有相应的步骤，在作业里也有相应的作业项。在转换里的“XSD Validator（XSD验证）”通过 XSL 转换来验证数据流里的 XML 格式的数据，作业里的“XSD Validator（XSD验证）”是验证一个完整的XML文件。如果想更深入了解 XSLT，包括它的一些例子，参考http://en.wikipedia.org/wiki/XSL_Transformations。可以把这个页面上的例子保存在本地，然后用Kettle的XSL 转换步骤测试一下。

第21章详细讲解了XML格式，以及读写XML数据的方式。

特殊文件类型

还有一些介于文本文件和数据库之间的文件，像数据库但又不是数据库。这些文件本来不会引起太多问题：但如果人们把这些文件当真正的数据库来使用，容易产生一些问题。Kettle 里有一些步骤来处理这类文件，这些步骤如下。

- **Access 输入步骤**：很多公司都使用Access文件。Access文件是一种轻量级的原型数据库或个人数据库，在被增加网络驱动功能后，它可以作为一些紧急任务的解决方法。Kettle 里有“Access 输入”步骤，可以从Access文件里抽取数据，但要注意，如果要让Kettle 访问Access，Access不能被加密（因为在Kettle里不能输入用户名和密码）。另外Kettle不支持较早的Access版本，如Access 2000。如果要访问这些较早的版本或者要让

Kettle 访问加密的Access，要在“表输入步骤”里使用ODBC的方式连接。

- **XBase输入步骤**：时间回退到上世纪80年代和90年代初，dBaseⅢ、Ⅳ、Ⅴ是非常流行的PC机上的数据库。Kettle 提供了“XBase输入步骤”以支持从这些文件读取数据，这个步骤在科学研究领域是非常有用的，因为在这个领域，很多数据还是保存在DBF文件中的。
- **Excel文件输入**：你应该想尽一切办法避免把 Excel文件作为输入数据源。如果你不得不使用Excel 作为数据源，Kettle也提供了和“文本文件输入”步骤类似的“Excel文件输入”步骤。

另外，对于其他的一些特殊数据源，如LDAP、LDIF、ESRI Shape文件和属性文件，请参考在线帮助文档。

6.1.2 数据库抽取

一说数据库，人们一般想到是关系型数据库系统（RDBMS），例如Oracle、SQL Server或MySQL。我们这里说的数据库抽取，也是指的这些数据库。但目前数据库领域发生了很多新的变化，最重要的就是no-sql或not only SQL数据库的出现，例如Hadoop、Hypertable、CouchDB或Amazon SimpleDB等。这类数据库参考 <http://nosql-database.org/>，我们本节的讨论，还是集中在传统的关系型数据库，从“表输入”步骤开始。尽管这个步骤在在线帮助文档中已经解释得很详细了，但我们下面的案例里还是要解释一下这个步骤里的参数和变量替换如何工作。

例子：创建参数查询

我们还用第4章的sakila数据库的例子。实际上第4章的“load_dim_staf”转换里已经使用了参数查询。总结一下，可以有两种参数化的查询方法：使用变量替换和使用参数。第4章就是使用参数的方法，这种方法需要在“表输入”步骤的前面有一个步骤，用来给“表输入”步骤提供一个或多个参数，这些参数替换“表输入”步骤的SQL语句里的问号。这种方法的主要配置窗口如图6-5所示。

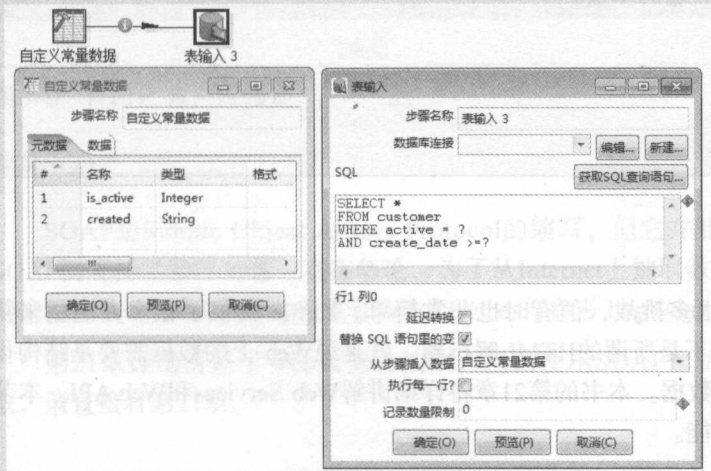


图6-5 参数化查询

上面例子中的“自定义常量数据”步骤只用来演示，在实际使用中，要有其他步骤替换这个步骤。在本章后面的CDC部分，我们还能看到一个类似的例子。

第二种方法要使用变量，变量要在使用变量的转换的前面进行某个转换设置。设置变量的转换如图6-6所示，设置变量的转换往往是作业里的第一个转换。

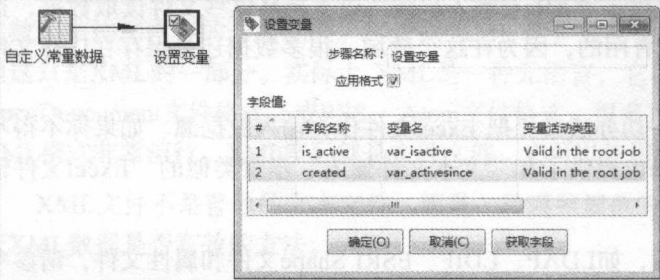


图6-6 设置变量的转换

在后面转换的表输入步骤里可以使用这些变量，查询里的变量名会被变量的值替换，使用变量的表输入步骤如图6-7所示。

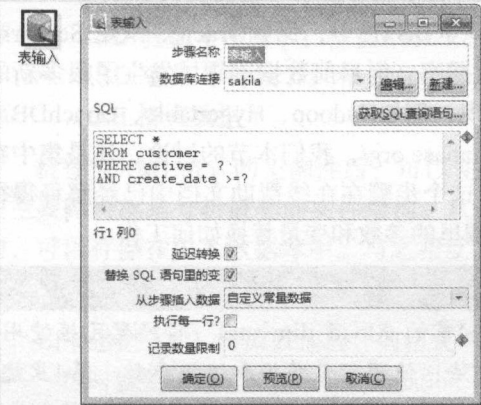


图6-7 使用变量的表输入步骤

上面的两个转换都在一个作业里，图6-8显示了这两个转换，第一个转换是设置变量，第二个转换是使用变量作为表输入步骤的参数。



图6-8 使用变量的作业

6.1.3 Web数据抽取

从Web 上获取数据往往面临很多挑战，但有时也非常简单。Kettle 提供了很多方法用来从Web获取数据，请注意这些方法并不是所谓的HTML解析器。就是说Web 上的数据需要是结构化的数据，Kettle 转换才能读取这些数据。本书的第21章将详细讲解Web Services和Web API。本节主要介绍几个Web数据源的基本功能。

基于文本的Web数据抽取

文本文件可以存储在Web服务器上，并通过HTTP的方式访问。在这种情况下，使用 Kettle 来获取这些数据就是非常容易的事情了。因为 Kettle 使用了Apache VFS系统，VFS系统可以像处

理本地文件一样处理HTTP文件，所以在“文本文件输入”步骤中可以直接把URL作为文件名。下面的URL就是Web文本文件的例子，这个文件保存了ISO 3166 标准的国家名称和编码：<http://www.ip2location.com/download/iso3166.txt>。

HTTP 客户端

从Web上抽取结构化文本文件的另一个方法是使用“HTTP client（HTTP 客户端）”步骤。这是一种抽取Web数据的简单方法，它调用URL，并返回一个字符串作为结果。返回的字符串是用分隔符分割的文本或是XML格式的字符串，可以用XML输入步骤做进一步处理。在我们下面的例子中可以看看 HTTP 客户端的各种选项的功能，我们使用 GeoNames 网站作为例子<http://ws.geonames.org>。我们使用“HTTP 客户端”步骤从这个网站抽取荷兰的国家信息。

首先，创建一个新的转换并添加一个“生成记录”步骤。因为“HTTP客户端”只是一个查询类步骤，它需要一个输入类步骤激活它，如果没有输入类步骤，查询类步骤不会做任何事情。在“生成记录”步骤里设置记录数为1，并设置一个类型为 String的字段 countryurl，这个字段的值应设置为我们要抽取数据的URL，可以用下面的值：

- <http://ws.geonames.org/countryInfo> 抽取所有国家信息，保存为XML格式字符串。
- <http://ws.geonames.org/countryInfoCSV>抽取所有国家信息，保存为CSV格式字符串。
- 通过参数只抽取一个国家的信息，如<http://ws.geonames.org/countryInfoCSV?country=NL> 将得到荷兰的信息，保存为CSV格式字符串。

然后，添加一个“HTTP 客户端”步骤，并把这两个步骤连起来。对于“HTTP客户端”来说，选择“Accept URL from field”选项，并选择 countryurl作为URL的来源字段。现在，可以预览一下，我们能看到如图6-9所示的结果。

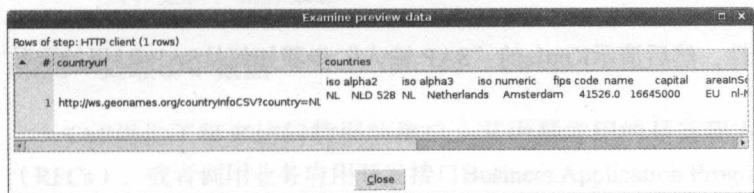


图6-9 HTTP 客户端预览

使用SOAP

SOAP是Simple Object Access Protocol的缩写，但它并非像它的名字这么简单。历史上，SOAP的支持者和反对者一直在争论，为了从Internet上抽取数据，我们也只支持SOAP的最基本的功能。关于SOAP的背景信息和Web Services可参考 http://en.wikipedia.org/wiki/web_service。

第21章详细讲解了Web数据的抽取，包括使用SOAP，所以如果想更深入了解Web数据抽取，请直接看第21章。

6.1.4 基于流的和实时的抽取

目前，数据集成工作变得越来越实时化，就是说数据的发生和在报表或分析中看到这些数据的间隔时间变得越来越短。尽管大多数人还把Kettle看作是面向批处理的ETL工具，但Kettle实

际上是和流程类型无关的工具。在第18章,我们演示如何使用Kettle从Twitter中抽取流数据,在第22章,我们演示如何使用Kettle抽取实时数据并生成Pentaho 报表。

6.2 处理ERP和CRM系统

ERP是Enterprise Resource Planning的缩写,它是为企业的业务流程提供全部支撑的系统,从人力资源到采购、生产、货运、发票等。ERP系统的历史还要从30年前开始,第一个生产资源计划系统Manufacturing Requirement Planning (MRP)出现了,用来支持企业的生产过程。后来第一代的MRP系统被第二代应用更广泛的MRP II系统替代,但第二代也还是只限于生产范围内。随着系统的不断成熟,开始覆盖到业务流程里的更多范围,包括人力资源、财务和客户管理,MRP里的M其实已经显得过时了,所以M被E (Enterprise)替代。这样可以说明任何企业都可以使用ERP系统,而非生产制造类企业。

在过去十年里,出现了很多MRP和ERP软件提供商。最有名的商业解决方案,也占有最大市场份额的就是SAP/R3、Oracle E-Business Suite和Microsoft Dynamics AX,前两个解决方案一般面向大型的跨国公司,后面的解决方案面向SMB (Small and Medium Business),即中小型企业。但对于大多数公司来说,ERP提供的是广度上的解决方案,而非深度上的解决方案,于是很多公司,如Siebel和PeopleSoft (现在都是Oracle的)、Salesforce.com和Amdocs等提供某些特定领域,如人力资源领域、CRM系统或销售支持系统的深度的解决方案。

因为商业软件价格昂贵,使用复杂,开源的替代品随之出现了。现在像OpenBravo、OpenERP, Adempiere和TinyERP这样的开源ERP软件可以在某些程度上和高价的商业软件竞争。在销售支持、CRM这一领域,最大的开源软件是SugarCRM,它可以同时提供开源社区版和商业企业版。

本节先说明ERP数据的特殊性,然后演示Kettle的“SAP 输入”步骤如何从SAP/R3里抽取数据。

6.2.1 ERP 挑战

像SAP, J/D Edwards, Lawson 这样的ERP系统,都有两个相同点:数据库中有大量的数据库表,以及数据都是以编码的方式保存。这样直接访问这些数据库表将非常困难。除此之外,表名和列名也都以编码加密的方法命名,如f4211 实际是Sales Order。在这些系统里,SAP尤其复杂,而且软件的License条款也禁止直接访问数据库。目前,一个标准的SAP程序包含了超过 70 000 个表而且大多数的逻辑和元数据都在应用里实现而不是在数据库里实现。所以从SAP里获取信息的唯一方式就是通过SAP自己提供的接口。几乎所有的ERP软件提供商都会提供类似图6-10那样的一种解决方案,它们提供了一个抽象层来访问底层数据。使用抽象层有下面几个优点:

- 不必知道底层的数据库结构。
- 容易找到可用的组件或功能。
- 软件供应商可以随时改变数据库的结构,而接口可以不受影响。

当然这种方法也有一定的成本:主要是性能问题。调用API的过程,需要先查询元数据,然后再根据元数据翻译成访问数据库的查询,这些过程都要花费时间。但这种访问方式是唯一的

访问方式，所以在抽取数据之前要明确抽取数据所用的时间能否满足ETL时间窗口的要求。

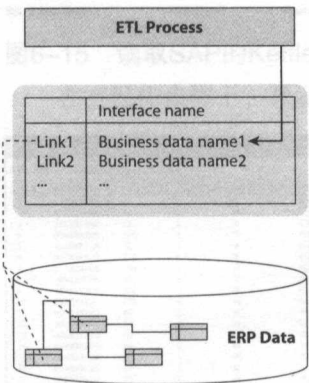


图6-10 业务数据层

6.2.2 Kettle ERP插件

因为Kettle 提供了插件架构，目前有几个商业化的第三方插件可以用来从SAP或SugarCRM里抽取数据。在Kettle 4 里，Pentaho 社区也开发了几个开源的ERP/CRM插件，在编写本书时，Kettle可以提供下面几种插件。

- “SAP Input” 步骤：Kettle可以通过SAP接口从SAP系统里抽取数据。
- “SalesForce input” 步骤：Kettle可以通过标准的Web Services调用从SalesForce里抽取数据。
- “SalesForce output” 步骤：Kettle可以通过标准的Web Services调用，向SalesForce里插入、更新、删除数据。

6.2.3 处理SAP数据

SAP提供了很多访问数据的接口，其中最常用的是远程方法调用Remote Function Calls (RFCs)，或者调用业务应用开发接口Business Application Programming Interface (BAPI)。澳大利亚的Pentaho合作伙伴Aschauer EDV 开发了开源的Kettle插件，可以通过这两种方式获取SAP数据。下面的例子演示了如何使用SAP插件从 SAP/R3的财务模块里抽取Billing Document的数据。

说明：作者在这里感谢来自Aschauer EDV的Bernd Aschauer和Robert Wintner，他们提供了这部分使用的截图。

在使用SAP输入步骤之前，需要先从SAP网站下载SAP Java Connector文件（sapjco3.jar或类似的包），并把这个jar包复制到Kettle的libext 目录下。然后，把“SAP Input”步骤拖曳到画布上，并在数据库连接里创建一个SAP 连接。一个SAP连接需要一个服务器地址、用户名和密码。图6-11 显示了设置好参数的连接例子。

如图6-11所示，每个SAP实例都有一个系统号，为了建立连接还要提供客户端ID和语言等等。这和使用标准的SAP客户端连接时需要的信息是一样的。输入完必要的连接信息后，单击“Test”按钮，测试是否可以连接成功。

建立完连接后，在“SAP Input”步骤的编辑窗口里单击“Find it”按钮打开函数浏览窗口，

可以使用正则表达式查找函数名，如图6-12所示。

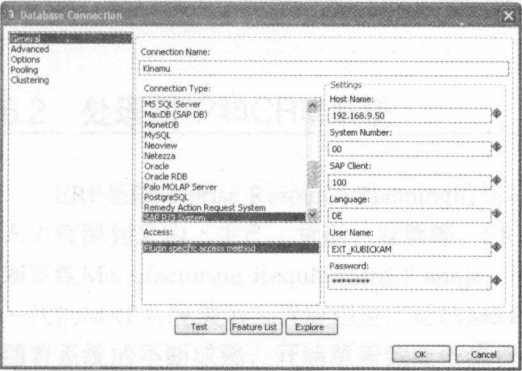


图6-11 创建SAP/R3 连接



图6-12 SAP函数浏览器

每个选中的函数一般都有多个输入和输出字段，在选中需要的函数后，在“SAP Input”步骤里会出现这些输入和输出字段，如图6-13所示。

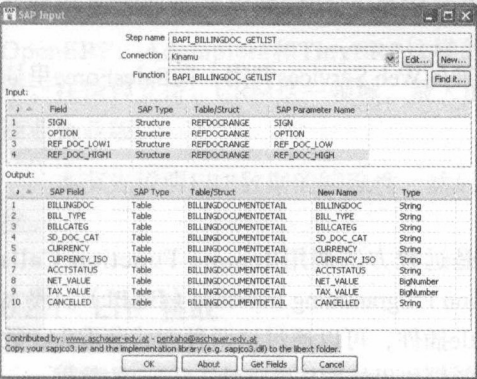


图6-13 SAP Input步骤

输入字段的数据需要在“SAP Input”步骤之前通过“Generate Rows”或“Data Grid”，或者通过数据库参数表来设置。这个例子里需要输入一行四列字符串数据，如图6-14所示。

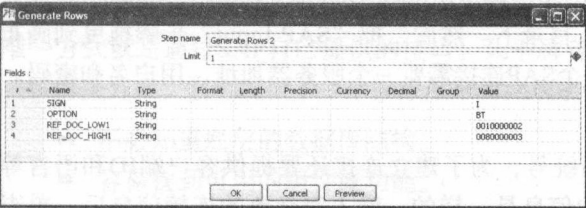


图6-14 “Generate Rows” 步骤为 “SAP Input” 步骤提供参数

这个例子的完整转换，也非常简单。图6-15演示了转换里的三个步骤，设置参数值、抽取SAP数据、把数据放到空操作插件中。

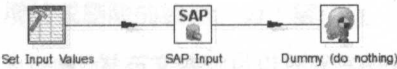


图6-15 读取SAP的Kettle转换

在空操作步骤上，预览数据，可以看到如图6-16所示的预览结果。

Rows of step: Dummy (do nothing) 2 (66 rows)

| # | BILLINGDOC | BILLTYPE | BILLCATEG | SD_DOC_CAT | CURRENCY | CURRENCY_ISO | ACCTSTATUS | NET_VALUE | TAX_VALUE | CANCELLED |
|----|------------|----------|-----------|------------|----------|--------------|------------|-----------|-----------|-----------|
| 1 | 0900000000 | F2 | L | M | EUR | EUR | C | 45 | 9 | |
| 2 | 0900000001 | F2 | L | M | EUR | EUR | E | 11350.5 | 0 | X |
| 3 | 0900000002 | S1 | L | N | EUR | EUR | E | 11350.5 | 0 | |
| 4 | 0900000025 | ZTE | L | M | EUR | EUR | E | 226.2 | 45.24 | X |
| 5 | 0900000026 | S1 | L | N | EUR | EUR | E | 226.2 | 45.24 | |
| 6 | 0900000027 | ZTE | L | M | EUR | EUR | E | 226.2 | 45.24 | |
| 7 | 0900000025 | ZTE | L | M | EUR | EUR | E | 226.2 | 45.24 | X |
| 8 | 0900000026 | S1 | L | N | EUR | EUR | E | 226.2 | 45.24 | |
| 9 | 0900000027 | ZTE | L | M | EUR | EUR | E | 226.2 | 45.24 | |
| 10 | 0900000030 | ZTE | L | M | EUR | EUR | C | 38.16 | 7.63 | |
| 11 | 0900000000 | F2 | L | M | EUR | EUR | C | 45 | 9 | |
| 12 | 0084000000 | F8 | L | U | EUR | EUR | D | 0 | 0 | |
| 13 | 0084000001 | F8 | L | U | EUR | EUR | D | 13 | 0 | |
| 14 | 0900000004 | F2 | L | M | EUR | EUR | G | 600 | 0 | |
| 15 | 0900000005 | F2 | L | M | EUR | EUR | C | 30 | 6 | X |
| 16 | 0900000006 | S1 | L | N | EUR | EUR | C | 30 | 6 | |
| 17 | 0900000007 | F2 | L | M | EUR | EUR | C | 30 | 6 | |
| 18 | 0900000008 | F2 | L | M | EUR | EUR | C | 0 | 0 | |
| 19 | 0900000037 | F2 | L | M | EUR | EUR | C | 105 | 0 | |
| 20 | 0900000038 | F2 | L | M | EUR | EUR | C | 105 | 0 | |
| 21 | 0900000013 | FAZ | P | M | EUR | EUR | C | 4460 | 892 | |
| 22 | 0900000014 | F2 | D | M | EUR | EUR | C | 19800 | 3960 | |
| 23 | 0900000015 | F2 | D | M | EUR | EUR | C | 2200 | 440 | |
| 24 | 0900000016 | FAZ | P | M | EUR | EUR | E | 4460 | 892 | X |
| 25 | 0900000017 | FA5 | P | N | EUR | EUR | E | 4460 | 892 | |
| 26 | 0900000018 | FAZ | P | M | EUR | EUR | C | 4460 | 892 | |
| 27 | 0900000019 | F2 | D | M | EUR | EUR | C | 20070 | 4014 | |
| 28 | 0900000020 | F2 | D | M | EUR | EUR | C | 2230 | 446 | |
| 29 | 0900000016 | FAZ | P | M | EUR | EUR | E | 4460 | 892 | X |
| 30 | 0900000017 | FA5 | P | N | EUR | EUR | E | 4460 | 892 | |
| 31 | 0900000018 | FAZ | P | M | EUR | EUR | C | 4460 | 892 | |

图6-16 SAP的Billingdoc数据

上面的第一个例子演示了如何使用标准函数获取直接可读的数据。另一个更通用的方法是使用RFC_READ_TABLE 函数，这种方法可以读取任意表的数据。但这种方法的缺点也很明显：需要知道要抽取数据的结构（至少要知道作业名），返回值是一列数据，需要手工分割，但是，这也是从SAP系统里获取数据的最灵活的方法。图6-17显示了在“SAP Input”步骤里如何通过 RFC_READ_TABLE 函数来抽取G/L Account Master表（也称为SKAT）的数据。SAP/R3 FI（财务）中其他类似的Master表还有KNB1（客户Master）和LFA1（供应商Master），从这些表名里还可以看出这个软件的德国血统：KN是Kunde（Customer）的缩写，LF是Lieferant（Vendor）。

Step name: RFC_READ_TABLE_2
Connection: Kinanu
Function: RFC_READ_TABLE

Input:

| # | Field | SAP Type | Table/Struct | SAP Parameter Name |
|---|-------------|----------|--------------|--------------------|
| 1 | DELIMITER | Single | | DELIMITER |
| 2 | QUERY_TABLE | Single | | QUERY_TABLE |
| 3 | ROWCOUNT | Single | | ROWCOUNT |
| 4 | ROWSKIPS | Single | | ROWSKIPS |
| 5 | FIELDNAME | Table | FIELDS | FIELDNAME |
| 6 | FILTER | Table | OPTIONS | TEXT |

Output:

| # | SAP Field | SAP Type | Table/Struct | New Name | Type |
|---|-----------|----------|--------------|----------|--------|
| 1 | WA | Table | DATA | WA | String |

Contributed by: www.sap-erp.com | info@sap-erp.com
Copy your sapclient and the implementation library (e.g. saptr03.00) to the libext folder.

OK About Get Fields Cancel

图6-17 使用RFC_READ_TABLE函数的“SAP Input”步骤

图6-17 里的输入输出的参数结构和第一个例子完全不同，在这里要指定分隔符、表名、要读取的行数、要跳过的行数、使用的字段名和过滤条件。从图中可以看到，字段名和过滤条件这两个字段是Table类型，也就是说这两个字段可以接收多个值。最后，为“SAP Input”步骤提

供数据的“Generate Rows”步骤如图6-18所示。

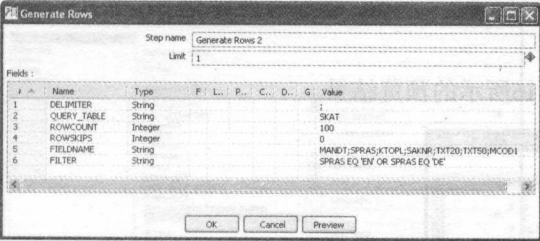


图6-18 SAP FC_READ_TABLE函数的输入规范

在使用“SAP Input”步骤的输出数据之前，还要拆分输出数据，拆分输出数据使用的分隔符和“Generate Rows”步骤定义的分隔符相同。使用Kettle的“Field splitter（拆分字段）”步骤可以很容易拆分输出列的数据，如图6-19所示。

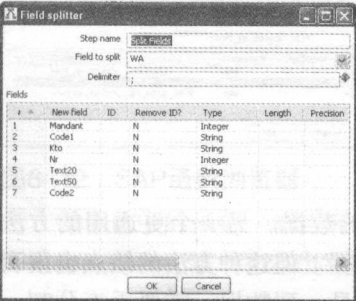


图6-19 使用“Field splitter”步骤把输出字段分割成多列

如果再仔细看图6-18和图6-19，可以看出字段名不同。因为要访问SAP系统里的字段，需要明确指定字段名。使用“Field splitter”步骤可以定义自己想要的字段名。唯一需要注意的是字段顺序和字段类型，而不是字段名。图6-20显示了使用 RFC_READ_TABLE函数抽取到的100行数据。

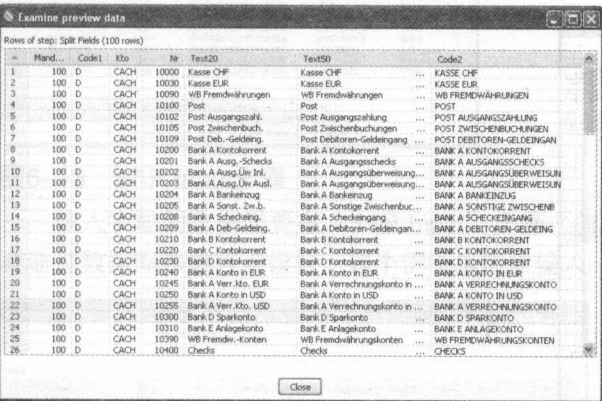


图6-20 使用 RFC_READ_TABLE抽取 G/L Account数据

6.2.4 ERP和CDC 问题

前面的SAP/R3系统的例子演示了如何从ERP系统中轻松地抽取数据。相信随着这方面需求的增多，会出现更多的ERP输入步骤。但使用这种方式有一个主要的问题，不能抽取变化的、新

增的或删除的数据。为了解决这一问题，可以采取下面三种方法中的一个。

- **发布文件：**可以要求 SAP 开发人员开发一个程序，这个程序可以把变化的数据发布到一个文本文件中。大多数项目都是这样实现的，因为ETL不能直接访问SAP系统。
- **基于快照：**在本章后面的基于快照的CDC技术的介绍中，将讲述如果使用快照和Kettle的“Merge Join”步骤来检查数据集的变化。
- **给RFC增加参数：**我们也可以给RFC增加一个参数，来过滤出编号的数据，但这需要在SAP上来做修改。

6.3 数据剖析

数据剖析是指从不同源系统中搜集数据的统计信息或其他相关信息的过程。通过数据剖析获得的统计信息对后面的ETL和数据仓库的设计过程都非常重要。数据剖析对数据质量来讲也很重要。在做数据清洗，提高数据质量工作之前，我们要知道当前数据的情况，以作为以后对比的基线版本。数据剖析可以完成下面3个层次的工作。

- **列特征分析：**搜集某一列数据的统计信息。
- **依赖性特征分析：**分析表中不同列之间的依赖关系。
- **连接特征分析：**分析不同表之间的依赖关系。

数据剖析工作的起点一般就是列特征分析，它可以生成关于一系列数据的统计数据，统计信息包括但不限于下面的信息。

- **不同值的个数：**这一列里有多少个值不相同的数据。
- **NULL值和空字符串的个数：**这一列里有多少个NULL值和空值。
- **最大最小值：**这个统计不只用于数值类型的数据，也可以用于文本类型的数据。
- **数值类型的合计、中位数、平均数、标准差：**各种关于数值类型的统计数据。
- **字符串模式和长度：**可以用来验证数据格式是否正确（如德国的邮政编码应该是五位数字）。
- **单词个数、大小写字符的个数：**这一列里单词的总个数，这些单词是全大写的还是全小写的，或者大小写混合的。
- **词频统计：**一列里，词频最大、最小的N个单词。

大多数的数据剖析工具都可以提供上面的这些统计信息，或者提供更多的信息。如果你要分析表内数据之间的依赖关系，或表间数据之间的依赖关系，情况就会变得比较棘手。例如分析邮政编码和城市、城市和区域、区域和国家之间的关系。城市名依赖于邮政编码，区域名依赖于城市，国家依赖于区域名。这些依赖关系违背了第三范式，如果在遵照第三范式的数据库系统中发现了这种依赖关系，就要非常小心，尤其在地址信息的处理上。有时候依赖关系并不是非常清晰，甚至令人迷惑，这样就不容易找出正确或不正确的数据。正因为这个原因，出现了很多地址匹配和清洗的解决方案和工具。例如，城市和区域的匹配：在美国至少有10个州下面都有一个叫 Hillsboro的城市。如果没有国家或邮政编码等信息，很难发现一条记录是否包含错误的数据。所以，我们需要一些外部的信息来验证自己的数据。

分析表间数据之间的关系要容易一些；只要看表之间数据列的依赖关系。在一个订单系统里，订单里的客户ID，肯定存在于客户表中。我们还可以做相反的关系验证，找出客户表中客

户ID有多少不在订单表里。这些表间依赖的验证同样适用于产品表、订单详细信息表, 库存表和供应商表, 等等。

使用 eobjects.org的DataCleaner

目前Kettle社区版里没有数据剖析的功能, Kettle使用eobjects.org开发的开源的 DataCleaner工具做数据剖析的工作。我们可以从<http://datacleaner.eobjects.org/>下载这个软件。非常容易安装, 在Windows下直接解压缩, 并执行 datacleaner.exe文件。在Linux下, 解压缩tar.gz文件, 将datacleaner.sh脚本改成可执行, 然后再执行这个脚本就可以了。如果想下次直接执行可以在Windows下创建一个快捷方式或在一个在Linux GNOME桌面环境下创建一个快捷启动。(在Kettle 4.4 里DataCleaner已经作为Kettle的一个视图插件, 可以集成到Kettle Spoon中——译者注。)

DataCleaner可以完成下面几个主要功能。

- **数据剖析:** 上面描述的所有数据剖析的功能。我们用这个功能可以随时了解数据库的数据状态。
- **数据验证:** 创建验证规则, 并验证数据是否符合规则。数据验证规则可以转换(手工转换)成Kettle的验证步骤。验证器可以强制要求数据满足验证条件并监控到不满足规则的数据。
- **数据比较:** 比较不同模式、不同表里的数据是否一致。

从上面的描述可以看到, DataCleaner没有直接提供表间数据剖析的功能, 但后面可以看到, 可以使用一些间接方法来做这个工作。

下面说明 DataCleaner的基本使用方法, 首先建立数据库连接, 对每一个数据库类型, DataCleaner都提供了数据库驱动。DataCleaner启动后, 会打开新任务面板, 在新任务面板里选择下面三个选项中的一个: Profile、Validate和Compare。单击Profile 按钮将开始一个特征分析任务。可以看到有两个面板, 左侧的面板里显示 “No data selected”, 如图6-21所示。

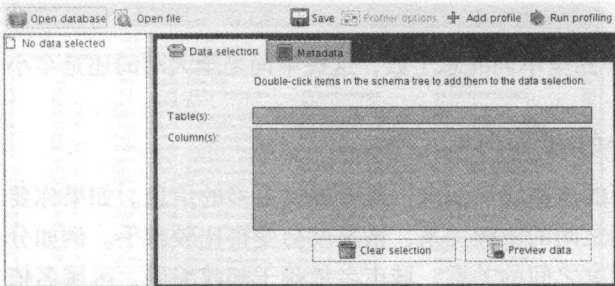


图6-21 特征分析任务

然后选择 “Open database”, 选中DataCleaner的 “sampledata” 这一项, 然后单击 “Connect to Database”。所有的字段此时都已设好。当单击左侧面板的加号, 打开 PUBLIC 树节点后, 显示出了数据库里的所有表。每个表还可以打开, 显示所有的列。双击某一列, 可以把这一列加到选中的数据里。可以看到表名填入到了Table(s)字段里, 列名填入到了Column(s)字段里。要从选中的数据里移去选中的列, 再双击或者使用 “Clear selection” 功能。“Preview data” 功能可以显示数据里的样本数据, 可以自己调整样本行的数量。每个选中的表的数据行会在一个窗口里显示出来。

和“Data selection”标签挨着的是“Metadata”标签。这个标签里显示所有选中的列的元数据。包括字段类型、字段长度、是否可以为空等元数据的描述信息，是否可以为空这个属性可以给用户对字段的第一印象。

添加特征分析任务

选完要做特征分析的列以后，就要添加一些特征分析的任务了。DataCleaner包含下面一些数据剖析功能。

- **标准度量**：行数，NULL值和空值个数，最大值，最小值。
- **字符串分析**：大小写字母所占的百分比，非字母字符所占的百分比，列里的最大单词个数，最小单词个数，单词总个数，字母总个数。
- **时间分析**：最小和最大的日期值，每一年包含的记录条数。
- **数值分析**：最大，最小，求和，算术平均，几何平均，方差，标准差。
- **生成字符串模式**：找出并且统计出字符串列的字符串模式。一般用于电话号码、邮政编码，或其他按照一定规范编码的字母数字格式的字符串。字符串格式的例子如 9999 aa（四个数字、一个空格、两个字母）、aaa-999（三个字母、连字符、三个数字）。
- **字典匹配**：验证选中的列里的数据是否包含在其他文件或数据库里（字典）。
- **正则表达式匹配**：验证选中的列里的数据是否符合给定的正则表达式。
- **日期掩码匹配**：使用日期掩码来匹配文本；不能用于日期类型字段，只能用于字符串类型字段，使用字符串来表示日期和时间。
- **数值分布**：前N和后N的词频统计（N为0~50，默认是5），单词出现的次数和百分比的排名。

可以很灵活地组织这些分析任务；每个任务都可以被保存，但只能保存任务和数据库连接，不能保存特征分析结果。可以把特征分析结果导出成XML格式的文件，但只能导出，不能再把结果导入进来。特征分析结果的持久化是以后版本的功能。

添加数据库连接

数据特征环境建好后的第一个工作就是添加正确的数据库驱动，并把数据库连接保存起来以便以后使用。第一个工作很简单，在DataCleaner窗口里，选择File→Register database driver。有两种方法添加一个新的驱动。第一种方法是自动下载并安装驱动。MySQL、PostgreSQL、SQL Server/Sybase、Derby和SQLite可以使用这种方法。第二种方法是手工注册.jar文件。为了帮助你找到正确的驱动文件，DataCleaner包含了大多数常用数据库驱动下载的网站，如Oracle、IBM DB2。下载完驱动后，选择刚下载完的驱动文件和里面的驱动类。对MySQL，可以使用自动下载和自动安装选项。

说明：如果你已经安装了MySQL JDBC 驱动，不必再下载，注册你自己的.jar文件即可。

要把数据库连接添加到“Open Database”对话框的数据库连接下拉列表里有一些复杂。需要修改DataCleaner的配置文件，配置文件在DataCleaner目录下，文件名是datacleaner-config.xml。使用能正确显示XML文件格式的文本编辑器，如Notepad++，打开这个配置文件，找到下面这行：

```
<!-- Named connections. Add your own connections here. -->.
```

这行的下面一项是下拉菜单里显示的空选项，保留不动。再下面一项是样本数据库的连接

选项，把这个选项里从<bean>到 </bean>之间的内容复制下来，并粘贴到这项的后面。调整刚粘贴的文本里的连接参数。下面的代码是sakila数据库的连接设置：

```
<bean class="dk.eobjects.datacleaner.gui.model.NamedConnection">
  <property name="name" value="sakila database" />
  <property name="connectionString"
value="jdbc:mysql://localhost:3306" />
  <property name="username" value="sakila" />
  <property name="password" value="sakila" />
  <property name="tableTypes">
    <list>
      <value>TABLE</value>
    </list>
  </property>
</bean>
```

为了能连到数据库，在password属性行的下面还要加一行：

```
<property name="catalog" value = "sakila" />
```

我们不推荐把密码保存在文本文件中，而且反对这种做法。在这个例子中可以把密码部分留空。然后每次创建 DataCleaner 任务的时候再提供密码。

在DataCleaner的在线文档里还有其他数据库连接的例子可供参考。

做最基本的数据剖析

DataCleaner优化了很多数据剖析的操作，你可以轻而易举地获得直观的剖析结果。在做数据剖析的时候，你只要反复单击剖析任务窗口右上角的“Add Profile”按钮，这样可以新建标准的度量分析或字符串分析、数值分析、时间分析等剖析功能。你可以在数据库的所有列上来完成这些最基本的剖析功能。

使用正则表达式

正则表达式是一种描述数据的方法，主要用于验证目的，也用于从文本中寻找特定的字符串。DataCleaner 在数据剖析里有正则表达式匹配，在数据验证里有正则表达式验证。在我们使用正则表达式之前，要把它添加到DataCleaner的正则表达式目录里。初始时，这个目录是空的，往里面添加正则表达式很容易。单击“New regex”会出现3个选项。第一个选项是手工创建一个新的正则表达式，最后一个选项是从.properties文件里获得正则表达式。第二个选项是最有用的：当你选择“Import from the RegexSwap”，会打开一个在线的正则表达式仓库，可以从里面挑一个。也可以把你的正则表达式贡献到这个在线的仓库里，<http://datacleaner.eobjects.org/regexswap>。从在线仓库里导入正则表达式以后，可以再改名字，或再稍微修改一下正则表达式，另外还有一个选项，可以输入字符串来验证正则表达式。如果DataCleaner提供的正则表达式不满足你的需求，互联网上还有大量的正则表达式网站，如<http://regexlib.com>包含了美国电话号码和邮政编码的正则表达式。如果想学习正则表达式，<http://www.regular-expressions.info>是一个不错的网站。

对于使用正则表达式验证的简单例子来说，创建一个新的特征分析任务，连接到salila数据库，在“Data Selection”标签里选择Address 表，使用“Add profile”功能添加一个“Regex matcher”并保留“Integer”正则表达式来验证地址里的电话号码字段，如图6-22所示。

正则表达式也是Kettle 里处理字符串数据的强有力工具。我们看过一个例子，通过正则表达式查找一个目录下所有的.txt文件。第7章和第20章也有几个使用正则表达式的例子。

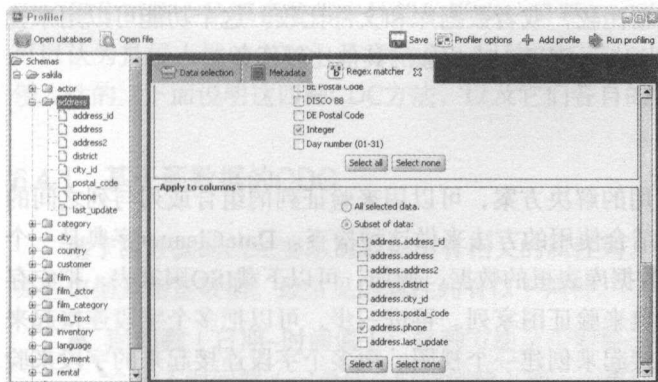


图6-22 电话号码的正则表达式匹配

浏览数据剖析结果

特征分析定义完成后，选择“Run profiling”开始分析。DataClenr 会显示一个状态窗口，可以监控分析处理过程。分析完成后，会出现结果标签，每个表一个结果，包含了所有分析列。图6-23 显示了上面例子里特征分析的结果。

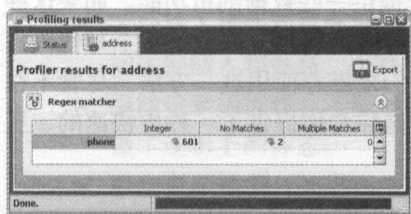


图6-23 数据剖析结果

可以清楚看到，有两个电话号码不符合正则表达式，此时可以看到DataCleaner的另一个强大功能：下钻到细节。单击两个发现的异常数据旁边的绿箭头，会打开如图6-24所示的窗口，显示出具体的数据行。

| address_id | address | address2 | district | city_id | postal_code | phone | last_update |
|------------|-----------------|----------|----------|---------|-------------|-------|-----------------|
| 1 | 47 MySahla D... | <null> | Alberta | 300 | -- | -- | 2006-02-15 0... |
| 2 | 28 MySQL Bou... | <null> | QLD | 576 | -- | -- | 2006-02-15 0... |

图6-24 数据剖析结果

但现在还没有完成。你单击右键，还可以发现有两个导出选项：一个是导出选中的数据，一个是导出全部表格。后一个选项也会把列头导出，第一个选项只导出数据。选中的单元格可以不相邻，如可以按住Ctrl键，选中地址，再选中电话号码列，并导出这两列到剪贴板中。最后，可以把剪贴板里的数据粘贴到电子表格或其他文件中。

验证和比较数据

验证和数据剖析功能非常类似，但多了一些功能。它可以检查NULL值或检查数据取值范围。另外还有JavaScript 等更高级的验证，可以使用Java表达式来验证数据。验证和数据剖析的

输出不同：验证只输出不满足条件的数据，以及不满足条件的记录数。DataCleaner 开发的路线图显示，将来可能把验证和数据剖析整合起来，统一对外提供服务。

数据比较功能是指不同数据源的数据比较，或者数据库和文件比较。这个功能可以用来检查是否rental 表里的所有客户都存在于customer 表里，或相似的其他比较任务。

通过字典实现列的依赖性检查

DataCleaner并没有提供一个即开即用的解决方案，可以用来验证列的组合或列与列之间的依赖关系。一般采用字典和数据库视图结合使用的方法来做这种检查。DataCleaner字典是一个文本文件，文本文件里的数据用来验证数据库表里的数据。例如，可以下载ISO国家表，把它存在文本文件里，把这个文本文件作为字典来验证国家列。再进一步，可以把多个字段连接起来保存在字典里，同时把要验证的数据连接起来创建一个视图。用多个字段连接起来的字段来验证这个视图，DataCleaner把不能和字典匹配的每个数据行都记录下来。也可以不使用文本文件作为字典，使用真正的数据字典表。字典表的数据库连接也需要先添加到DataCleaner的配置文件里，我们曾在前面的“添加数据库连接”里讲过。

替代的解决方案

其他可以做数据剖析的独立的开源软件或有此功能的开源软件非常少。我们介绍过的数据建模工具SQLPower也有一些基本的特征分析功能，Talend 也提供一些数据剖析功能。如果这些工具能满足你的需求，你只要使用它们就可以了。做数据剖析的另一种方法就是手工写代码。只有在做DataCleaner不能满足的非常特殊的特征分析时，我们才推荐使用这一方法。尽管不在本书范围内，但实际上我们可以自定义DataCleaner的功能，这样为你提供了一种更快速的、更可靠和更灵活的解决方案。

使用Kettle 进行文本特征分析

前面说过，Kettle并不提供和DataCleaner一样的特征分析功能。但对于文本类型的数据则例外。如果使用DataCleaner来做文本的特征分析，打开文件后，你会发现所有的类型都是Varchar，另外，DataCleaner只能识别逗号、分号、制表符作为分隔符，这样限制了这个工具的功能。在这些情况下，Kettle可以做文本类型的数据剖析。“Text file input（文本文件输入）”步骤的“Get Fields（获取字段）”的功能可以识别出字段的类型和长度。但Kettle 识别文本类型还有一些问题，如果文本字段的内容是以分隔符分隔的多个数字（如21;33;56）或以分隔符分隔的时间格式（如21:34），Kettle 会把它识别成长度为2的数字，而不是长度为8或5的字符串。这是一个已知问题，在Kettle 4.1中会解决。

6.4 CDC：变更数据捕获

ETL过程的第一步就是从不同的数据源抽取数据并把数据存储在数据缓存区。这个过程的主要挑战就是初始加载的数据量大和比较慢的网络延迟。在初始加载完成后，不能再把所有数据重新加载一遍。我们只需要抽取变化的数据。识别出变化的数据并只抽取这些变化的数据称为变化数据捕获（Change Data Capture）或CDC。

说明：本节的部分内容已经在 *Pentaho Solutions* 一书中讲述过。

基本上，CDC可以分为两种，一种是侵入性的，另一种是非侵入性的。所谓侵入性的是指CDC操作可能会给源系统带来性能的影响。只要CDC操作以任何一种方式执行了SQL语句，就可以认为是侵入性的CDC。不幸的是，四种CDC方法中的有三种都是侵入性的，只有一种不是侵入性的。下面说明这四种CDC方法，以及它们各自的优缺点。

6.4.1 基于源数据的CDC

基于源数据的CDC要求源数据里有相关的属性列，ETL过程可以利用这些属性列，来判断出哪些数据是增量数据。最常见的属性列有以下两种。

- **时间戳（日期-时间值）：**这种方法至少需要一个更新时间戳，但最好有两个时间戳：一个插入时间戳（记录什么时候创建）和一个更新时间戳（记录什么时间最后一次更新）。
- **序列：**大多数数据库都有自增序列。如果数据库表用到了这种序列，就可以很容易识别出新插入的数据。

这两种方法都需要一个额外的数据库表来存储上一次更新时间或上次抽取的最后一个序列号。在实践中，一般是在一个独立的模式下或在数据缓冲区里创建这个参数表，不能在数据仓库里创建，更不能在数据集市里创建。基于时间戳和自增列的方法是CDC最简单的实现方式，所以也是最常用的方法。但它的缺点也是很明显的，主要如下。

- **区分插入操作和更新操作：**只有当源系统包含了插入时间戳和更新时间戳两个字段，才能区别插入和更新，否则不能区分。
- **删除记录的操作：**不能捕获到删除操作，除非是逻辑删除，即记录没有真的删除，只是做了逻辑上的标志。
- **多次更新监测：**如果在一次同步周期内，数据被更新了多次，只能同步最后一次更新操作，中间的更新操作都丢失了。
- **实时能力：**时间戳和基于序列的数据抽取一般适用于批量操作，不适合于实时场景下的数据加载。

在Kettle里使用时间戳方式的CDC：例子

我们使用第4章提供的sakila数据库演示基于时间戳的CDC，sakila数据库里所有的表都有更新时间戳，Customer表甚至还有插入时间戳。因为Customer表可以区分插入时间和更新时间，所以我们的例子就使用了Customer表。另外，我们还需要把上一次加载时间存储在属性文件里或参数表里。在我们的例子里，在sakila_dwh库里创建了一个cdc_time表，这个表里有两个字段，一个是last_load时间戳字段，一个是current_load时间戳字段。最开始，这两个时间戳字段都设置成一个很早以前的时间（当开始加载时，current_load时间戳设置为当前的时间）。

首先，创建这个时间戳表。

```
CREATE TABLE `cdc_time` (
  `last_load` datetime,
  `current_load` datetime)
```

然后插入默认值：

```
insert into cdc_time values ('1971-01-01 00:00:01', '1971-01-01 00:00:01')
```

该表的逻辑描述如下：

1. 加载作业开始后，作业要先把current_load 时间设置成作业的开始时间。可以通过“Get System Info step”完成这一功能，在这个步骤里创建一个“system date (fixed)”类型的字段，字段名是sysdate。然后再创建一个“Insert / Update ”步骤，把“Get System Info”步骤和“Insert / Update (插入/更新)”连接起来。在“Insert / Update”步骤的“Update fields”部分里，用流里的字段“sysdate”去更新表里的字段“current_load”。另外还要设置查询条件部分，把表的“current_load”的条件设置为“IS NOT NULL”即可。
2. 从Customer 表里抽取数据的查询语句使用开始日期和结束日期。查询条件类似下面的语句：

```
(create_date >= last_load AND create_date < current_load)
或：
```

```
(last_update >= last_load AND last_update < current_load)
```

这里我们需要两个表输入步骤，一个用来从 cdc_time 表中抽取时间，另一个从 Customer 表中抽取需要的数据。另外再看查询条件，可以发现last_load和current_load 分别出现两次。就是说在第一个表输入步骤中，这些时间值需要被抽取出来两次。

```
SELECT
last_load    last1
, current_load    curl
,      last_load    last2
, current_load    cur2
FROM cdc_time
```

在下一个步骤里，就是Customers的表输入步骤里，选中“Replace variables in script”，在“Insert data from step”下拉列表里选中上个表输入步骤。在SELECT语句里写入下面的查询条件：

```
WHERE
(create_date >= ? AND create_date < ?)
OR
(last_update >= ? AND last_update < ?)
```

前一个步骤传来的参数将替换上面语句里的问号，第一个问号的值是last1，第二个问号的值是curl，等等。

3. 通过比较create_date和last_update的值是否相等，可以判断出是新增的还是更改的数据。

```
CASE
WHEN create_date = last_update THEN 'new'
ELSE 'changed'
END AS flagfield
```
4. 如果转换中没有发生任何错误，要把 current_load字段里的值复制到 last_load字段里。如果转换中发生了错误，时间戳需要保持不变。把 current_load字段里的值复制到 last_load字段里需要“Execute SQL script”步骤，脚本如下：

```
update cdc_time set last_load = current_load
```

cdc_time表里之所以要有两个字段（尤其是current_load字段），是因为在加载过程中，会有新的数据被插入或更新，为避免脏读取和死锁的情况，最好给create和update时间戳设定一个上限条件。

6.4.2 基于触发器的CDC

当执行 INSERT、UPDATE、DELETE 这些 SQL 语句时，可以激活数据库里的触发器，并执行一些动作。就是说触发器可以用来捕获变更的数据并把数据保存到中间临时表里。然后这些变更的数据再从临时表里取出，被加载到数据仓库的数据缓存区里。但在大多数场合下，不允许向数据库里添加触发器（需要变动数据库，服务协议里不允许或者管理员不允许），而且这种方法还会降低系统的性能，所以这种方法一般用得不多。

作为直接在源数据库上建立触发器的替代方法，可以使用源数据库的复制功能，先把源数据库上的数据复制到数据仓库的接收表中，在接收表上建立触发器以提供 CDC 功能。尽管这种方法看上去过程冗余、需要额外的存储空间，但实际这种方法非常有效，而且没有侵入性。

复制功能是大部分数据库系统的标准功能，如 MySQL、PostgreSQL 和 Ingres 等。

基于触发器的 CDC 是最具有侵入性的方法，但也有优点，它可以监测到数据的所有变化，可以做到准实时的加载。缺点就是需要 DBA 的允许（因为要变动源系统），另外各个数据库的触发器的语法不同。

6.4.3 基于快照的CDC

如果没有时间戳、也不能用触发器，就要使用快照表了，通过比较来获得变化。快照表就是一次性抽取源系统中全部数据，把这些数据加载到数据仓库的缓冲区中。下一次需要同步时，再从源系统中抽取全部数据，并把全部数据也放到数据仓库的缓冲区中，作为这个表的第二个版本，然后再比较这两个版本的数据，找到变化。例如，两列的一个表，列名是 ID 和 Color。这个表的两个版本如图 6-25 所示。

| 快照1 | | 快照2 | |
|-----|-------|-----|--------|
| ID | COLOR | ID | COLOR |
| 1 | Black | 1 | Grey |
| 2 | Green | 2 | Green |
| 3 | Red | 3 | Blue |
| 4 | Blue | 4 | Yellow |

图6-25 快照版本

有多个方法可以获取这两个版本数据的差异。第一个方法就是在主键 ID 上做外连接并根据字段的比较结果增加一个标志字段（I 代表插入、U 代表更新、D 代表删除、N 代表没变化），另外过滤掉没有变化的记录，SQL 语句如下：

```
SELECT * FROM
(SELECT CASE
    WHEN t2.id IS NULL THEN 'D'
    WHEN t1.id IS NULL THEN 'I'
    WHEN t1.color <> t2.color THEN 'U'
    ELSE 'N'
END AS flag
CASE
    WHEN t2.id IS NULL THEN t1.id
    ELSE t2.id
END AS id
```

```
,          t2.color
FROM          snapshot_1 t1
FULL OUTER JOIN snapshot_2 t2
ON            t1.id = t2.id
) a
WHERE flag <> 'N'
```

当然，这样的SQL 语句需要数据库支持外连接，对于MySQL这样不支持外连接的数据库，可以使用类似下面的SQL语句：

```
SELECT 'U' AS flag, t2.id AS id, t2.color AS color
FROM snapshot_1 t1 INNER JOIN snapshot_2 t2 ON t1.id = t2.id
WHERE t1.color != t2.color
UNION ALL
SELECT 'D' AS flag, t1.id AS id, t1.color AS color
FROM snapshot_1 t1 LEFT JOIN snapshot_2 t2 ON t1.id = t2.id
WHERE t2.id is NULL
UNION ALL
SELECT 'I' AS flag, t2.id AS id, t2.color AS color
FROM snapshot_2 t2 LEFT JOIN snapshot_1 t1 ON t2.id = t1.id
WHERE t1.id IS NULL
```

这两个SQL 语句的结果都是一样的，结果如图6-26所示。

| Flag | ID | COLOR |
|------|----|--------|
| U | 1 | Grey |
| D | 3 | NULL |
| I | 5 | Yellow |

图6-26 快照比较结果

现在，大部分ETL工具都可以比较两个表之间的差异，并增加一个字段，使用 I、U、D代表各种变化，人们一般更喜欢使用这种ETL功能而不是SQL语句来比较两个表的差异。Kettle 里的“Merge rows”步骤也有这个功能。这个步骤读取两个使用关键字排序的输入数据流，并基于数据流里的关键字比较其他字段。可以选择要比较的字段，并设置一个标志字段，作为比较结果输出字段。我们用第4章的sakila数据库里的Customer 表做个例子，先把这个表里的全部数据抽取出来，再修改源表里的几条记录，然后就可以创建基于快照的CDC 转换。

1. 第一步先把Customer 表里的全部数据保存在另一个数据库里（使用第4章的sakila_dwh 数据库即可），也可以直接保存在 sakila 库里。在实际场景中，一般不允许直接保存在源数据库系统或数据仓库中。所以最好保存在 sakila_stg这类的缓冲数据库中。

说明：在数据库连接里选中支持 Boolean的选项，否则 Kettle 会把 valid字段设置为长度为 1 的字符类型。在数据库连接对话框的“Advanced” 标签下设置支持Boolean 类型。

在这个例子里，我们使用在第4章中已经创建的sakila_dwh数据库，我们把Customer 表里原来的数据保存为 customer_2 表。

2. 下一步要对原来的数据做一些修改。例如改变 last_name字段，使一个用户失效，添加一个新的用户，这些都要在源数据库系统上进行修改。另外为了演示Kettle 能检测出删除的数据，可以在 customer_2 表里增加一行，这行在源系统中不存在，这样可以模拟出在源系统中删除一条的情况。

3. 下面我们创建流程。创建两个“表输入”步骤，一个是sakila 库的表输入，另一个是 sakila_dwh 库的表输入。在表输入里选中所有字段，并按照关键字段排序。然后添加一个“Merge rows(diff)”步骤，把两个表输入步骤都连接到“Merge rows”步骤。打开“Merge”步骤，选择哪个步骤是旧数据来源，哪个步骤是新数据来源，选择标志字段（标志字段将包含 unchanged、changed、new和deleted数据），另外设置关键字段和需要比较的字段。该步骤配置如图6-27所示。

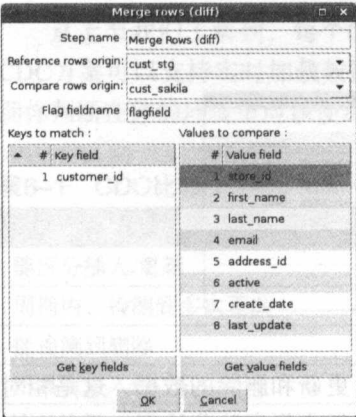


图6-27 “Merge rows” 步骤设置

4. 现在就可以预览“Merge rows”步骤的运行结果了。为了过滤没有发生变化的数据，我们需要在后面再增加一个“Filter rows”步骤，过滤条件是“flagfield=identical”，我们把所有没有变化的数据都发送到“dummy output”步骤，把新增、删除、修改的数据发送到步骤，如发送到“Copy rows to result”，可以留给后面的转换再继续处理，也可发送到“Output”里的“Synchronize after merge”步骤。Synchronize步骤可以根据标志字段自动进行增加、删除、修改等操作。配置窗口如图6-28所示。

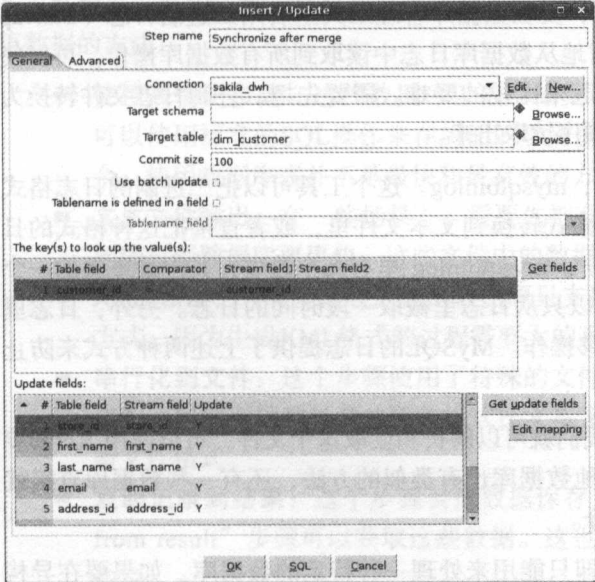


图6-28 Kettle 的“数据同步（Synchronize after merge）”步骤

完整的转换如图 6-29 所示，例子也可以从本网站下载（sakila_custdiff.ktr文件）。

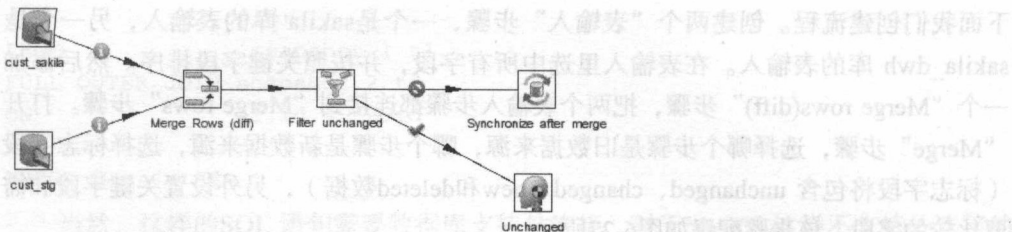


图6-29 基于快照的CDC方案

最后的运行结果如图6-30所示。

| # | customer_id | store_id | first_name | last_name | email | address_id | active | create_date | last_update | flagfield |
|---|-------------|----------|------------|-----------|---------------------|------------|--------|-------------------------|-------------------------|-----------|
| 1 | 1 | 1 | MARY | SMITH | MARY.SMITH@sakilac. | 5 | Y | 2006/02/14 22:04:36.000 | 2010/03/28 21:07:48.000 | changed |
| 2 | 2 | 1 | PATRICIA | JOHNSON | PATRICIA.JOHNSON@ | 6 | N | 2006/02/14 22:04:36.000 | 2010/03/28 22:07:48.000 | changed |
| 3 | 600 | 2 | MATT | CASTERS | MATT.CASTERS@pent | 605 | Y | 2010/03/28 21:12:10.000 | 2010/03/28 21:02:22.000 | new |
| 4 | 601 | 1 | ROLAND | BOUMAN | ROLAND.BOUMAN@p | 5 | Y | 2010/03/28 21:07:48.000 | 2010/03/28 21:07:48.000 | deleted |

图6-30 CDC 运行结果预览

从上面的例子可以看出，基于快照的CDC可以检测到插入、更新和删除的数据，这是相对于基于时间戳的CDC方案的优点，但它的缺点是要大量的存储空间来保存这些快照。另外，在表比较大时，也会有比较严重的性能问题。因为会有这种性能问题，所以我们前面演示了如何使用 SQL来做比较，数据库引擎的性能往往比ETL引擎的性能更好。第19章还介绍了使用 Kettle 在 Data Vault 模型下做CDC的例子。

6.4.4 基于日志的CDC

最高级的和最没有侵入性的CDC方法是基于日志的方式。数据库会把每个插入、更新、删除操作记录到日志里。如使用MySQL数据库，要在数据库管理工具里启用二进制日志（Startup variables → Logfiles）。启用后，可以准实时地从数据库日志中读取到所有数据库操作，可以使用这些操作来更新数据仓库中的数据。但做起来比说的要难。需要先把二进制日志文件转换为可以理解的格式，然后再把里面的操作按照顺序读取出来。

MySQL提供了一个可以读取日志的工具，mysqlbinlog。这个工具可以把二进制的日志格式转换为人类可以阅读的格式，然后可以把这种格式转换到文本文件里，或者直接把这种格式的日志导入到MySQL 客户端（用于 restore 操作）。mysqlbinlog 里有几个选项，其中最重要的一个选项就是可以设置开始/截止时间戳，这样可以只从日志里截取一段时间的日志。另外，日志里的每一项都有一个序列号，也可以用来做偏移操作。MySQL的日志提供了上述两种方式防止CDC过程发生重复和丢失数据的情况。

把mysqlbinlog的输出写到文本文件后，我们就可以解析和读取这个文件，可以使用 Kettle步骤来读取文件内容，并执行相应的语句。其他数据库也有类似的方法，还有一些数据库直接就提供了CDC机制，并作为数据库功能的一部分。

使用基于数据库的日志工具也有缺陷，即只能用来处理一种特定的数据库。如果要在异构的数据库环境下使用基于日志的CDC方法，就要使用类似Oracle GoldenGate或Attunity Stream 之类的商业软件。这些软件的一个主要缺点就是价格昂贵。不过目前市场上出现了一个很有意思

的开源替代品，Tungsten Replicator，它提供高级的主从复制选项，它可以支持基于语句的和基于数据行的复制服务，它是一个比二进制日志文件读取更高级的方案。在以后的发布版本中，它还会支持Oracle和PostgreSQL数据库，这样它就会成为一个非常有竞争力的产品。更多的信息参考：<http://www.continuent.com/community/tungsten-replicator>。

6.4.5 哪个CDC方案更适合你

从上节我们了解到，每个CDC方案都有自己的优缺点。有的CDC方案需要DBA同意，有的CDC方案可以支持实时加载数据，有的只能捕获到部分变更的数据。表6-1总结了一些方面，帮助你判断在你的环境里应该使用哪种CDC方案。

表6-1 CDC比较

| | 时间戳方式 | 快照方式 | 触发器方式 | 日志方式 |
|-------------|-------|------|-------|------|
| 能区分插入/更新 | 否 | 是 | 是 | 是 |
| 周期内，检测到多次更新 | 否 | 否 | 是 | 是 |
| 能检测到删除 | 否 | 是 | 是 | 是 |
| 不具有侵入性 | 否 | 否 | 否 | 是 |
| 支持实时 | 否 | 否 | 是 | 是 |
| 需要DBA | 否 | 否 | 是 | 是 |
| 不依赖数据库 | 是 | 是 | 否 | 否 |

6.5 发布数据

使用上面的方法获得变更数据后，要把这些数据发布到数据库中，Kettle 提供了几种方法发布这些数据，其中一种方法已经在基于快照的CDC例子中使用过了。下面列出其他几种发布变更数据的方法。

- **使用表输出步骤：**这些数据可以被存储到一个独立的缓存数据表里。这样做的好处是可以使用标准的SQL操作来在后面的步骤里再进行处理，缺点是会带来额外的负载和冗余。使用数据库表并不是最快和最有效的方法。
- **文本文件输出：**在一些场景下，需要先把变更数据输出到文本文件里，输出到文本文件比输出到数据库要更快。这些文件中的数据随后再被导入到其他系统中。
- **XML 输出：**只有在接收数据的数据源只支持 XML格式的数据时，才有必要用这种输出方式。因为生成XML格式的过程需要大的系统开销，所以尽量使用其他方式。
- **串行化到文件：**这个步骤使用了特殊的文件格式，只能被“De-serialize from file”步骤读取。但使用这种私有的文件格式比文本文件性能更好，因为Kettle 在读取时不用再解析文件格式。
- **复制记录到结果：**这个步骤会把数据保存在内存里，在后面的转换里使用“Get rows from result”步骤可以获取这些数据。这也是转换之间交换数据的最快方式，但只限于内存。

6.6 小结

本章介绍了如何使用 Kettle从不同的数据源获取数据。第一节主要介绍了抽取数据的四种方式:

- 基于文件的抽取, 使用平面文件和XML格式文件。
- 基于数据库的抽取, 在表输入里使用参数来抽取。
- 基于Web的抽取, 使用Web Services方式等。
- 流数据的抽取。

然后重点介绍了如何从 ERP和CRM系统中抽取数据, 通常是使用业务数据层或厂商提供的API, 因为在大部分情况下不能直接访问这些数据库。我们演示了如何使用 SAP 输入步骤从 SAP/R3 ERP系统中抽取数据。

后面, 我们介绍了数据剖析, 这是我们了解数据的结构和质量的一个重要方法。其中我们还介绍了eObject.org的DataCleaner, 这是一个开源的数据剖析工具, 它可以独立使用。

最后一节, 我们介绍了实现变更数据捕获(CDC)的几个方法, 包括:

- 基于时间戳的CDC
- 基于快照的CDC
- 基于触发器的CDC
- 基于日志的CDC

第7章 清洗和校验

对大多数用户来说，ETL的核心价值在T部分，T代表转换（Transform）。但实际上，这个阶段发生的很多工作，并不是一个T就能代替的。Ralph Kimball 把ETL称为ECCD，就是Extract、Cleanse、Conform和Deliver，Matt Caster使用Kettle这个名字给我们正在讲的这个ETL工具命名也是考虑到了这个单词有两个T，这两个T分别代表了传送（Transportation）和转换（Transformation）。本章主要讲解34个子系统中的关于数据清洗和验证的四个子系统，这四个子系统分别如下。

- 子系统4：数据清洗子系统
- 子系统5：错误处理子系统
- 子系统6：审计维度子系统
- 子系统7：排重子系统

本章所有例子都基于sakila数据库的customer和address表，只是把数据弄乱了一些，这样数据清洗步骤就可以做一些工作了，即在原始数据上面增加一些重复数据，再拼写错一些名字，再增加一些额外数据行。可以使用 sakilamods.sql脚本来做这些事情，这个脚本也可以从本书网站下载，当然也可以自己把数据改成自己想要的样子。我们后面也描述了清洗转换例子所要求的数据格式，可以按照要求自己构造数据。

本章很大一部分内容是数据验证，因为在清洗数据之前，必须明确哪些数据应该被清洗掉以及验证数据的条件。这里和第6章介绍的数据剖析的工作非常接近。数据剖析是理解数据质量和组成的第一步。数据剖析的结果可以被用来构建数据清洗和验证步骤的条件。

本章讲述的主题非常重要。2003年，数据仓库学院（Data Warehouse Institute，TDWI）估计数据质量问题每年会消耗6000亿美元或更多。

7.1 数据清洗

数据清洗是使用 ETL 工具最重要的原因之一。另一方面关于数据清洗工作也有很多争论, 如应该在 ETL 过程里做清洗工作, 还是应该在数据仓库里做清洗工作。我们在介绍什么是数据清洗之前, 先看看关于数据清洗的几个问题。

数据清洗是数据质量这一更大主题的一部分, 而数据质量又是数据管理这一主题的一部分。具有争议的是, 数据质量问题应该在根源处解决, 也就是事物型的业务系统里解决。如果这些数据在源系统里不能被清洗掉, 那只能在它上一级, 加载到数据仓库之前清洗掉, 这样带来的问题就是数据仓库里的数据和源系统里的数据不一样。Data Vault 模型的倡导者 (第 19 章) 在这个问题上有不同观点: 他们认为, 数据应该按照原样保存在数据仓库里, 只有在移动到数据集市里, 按照要求再做清洗。这样在加载时刻的数据都原封不动保存下来, 便于以后追踪。ETL 开发人员的工作量没有减少, 只是放到了数据处理环节的下游; 因为每个数据集市是单一的实体, 所以 ETL 开发人员的工作量可能还会增加。所以设计可复用的数据清洗转换是 ETL 开发过程中的一个重要部分。

Kettle 提供了很多步骤帮助你完成数据清洗工作, 无论从源系统中抽取数据清洗, 还是从数据仓库中抽取数据清洗。

注意: Arkady Maydanchik 提出了如下的数据质量的五个规则。

- 属性值域约束: 每个属性的取值范围域的约束。
- 关系完整性规则: 检查数据的唯一性和参照性约束, 这种约束可以从关系数据模型中得到。
- 历史数据规则: 对时间相关的数据, 要进行时间轴的约束, 值的模式匹配。
- 对象的状态依赖规则: 检验数据是否满足所谓的状态机模型的约束。
- 一般依赖规则: 描述复杂的属性的关系, 包括冗余、派生、部分依赖和相关性的属性。

详见 Arkady Maydanchik 的 *Data Quality Assessment* 一书, Technics Publications, LLC 出版社 2007。

如果从 Kettle 的角度来看 Maydanchik 提出的这五个分类, 第一个明显容易处理, 会在以后的章节深入介绍。引用完整性规则也可以使用查询类步骤来实现, 但是唯一完整性规则有些复杂。唯一完整性是指数据库里的每一条记录都指向现实世界中单一的、唯一的一个实体, 不能有两条记录指向同一个实体 (如一个人或一个产品)。本章的“数据排重”部分将介绍如何满足这一规则。除了前两个基本规则外, 后面的数据质量规则更复杂了, 因为没有标准的步骤来实现这些规则。但是使用 Kettle 完全可以处理这些规则。例如, sakila 数据库的状态依赖是指: 没有租赁就没有归还状态, 每一个租赁后面 (在某段时间内) 都应该有一个相应的归还状态或丢失赔款。租赁和归还之间的时间间隔是一个相关约束, 检查这个约束很容易。另外发货日期不能在订单日期之前, 收货日期不能在订单日期之前。

一般来说, 在 Kettle 里做依赖规则检查需要多个步骤, 我们后面可以看到地址校验例子。一个地址包含有多个列, 需要通过多个步骤来校验一个地址是否是城市里的有效地址, 邮政编码是否和城市名称匹配, 城市名称是否是国家或州 (省) 里的有效城市名称。

说明: 关于数据质量规则更详细的介绍以及如何评价数据质量, 我们推荐 Arkady Maydanchik 编写的 *Data Quality Assessment* 一书, Technics Publications, LLC 出版社, 2007。

另外还有一个比较好的资源, <http://www.dataqualitypro.com>, 这个网站里有很多数据质量方面的文章。

7.1.1 数据清洗步骤

Kettle里没有单一的数据清洗步骤,但有很多的步骤组合起来可以完成数据清洗的功能。数据清洗的工作从抽取数据时就开始了:很多输入步骤里都可以设置特定的数据格式,按照特定的数据格式来读取数据,尤其是日期和数值类型。

警告: 可以使用“表输入”步骤里的可自定义的SQL语句来做一些清洗工作,只抽取必要的数据库再传递给后面的步骤。但要注意,SQL语句太复杂会使以后的维护工作比较困难。我们也可以把数据原样读出来,然后使用Kettle的步骤转换,这里需要用户在效率和维护性之间平衡。

另外,避免使用自定义的SQL抽取数据还有一个原因:因为在进入Kettle时,数据就已经清洗了,所以ETL不能提供数据审计功能。大多数的Kettle都支持错误处理功能或条件处理功能来重新定向不满足一定条件的数据,见本章后面的“错误处理”部分。

转换目录下的步骤为清洗工作提供了很多不同的选项。一个功能很丰富的步骤就是“Calculator (计算器)”步骤,在前面的章节中也使用过这个步骤。我们不可能在这里列出这个步骤里的所有选项,因为版本的更新,里面的选项还在不断增长。我们这里只列出几个和数据清洗工作相关的选项。

- **ISO8601星期和年份数字:** 美国以外的很多国家使用不同的基于ISO8601标准的星期数字。不是所有的数据库都能从日期类型获得正确的星期和年份数据,所以计算器步骤里的这个功能很有用。
- **大小写转换:** 首字母大写、全部大写、全部小写可以让字符串的大小写统一。
- **返回/移除数字:** 这个功能可以把既包含数字也包含字母的字符串拆分成数字和字母部分。例如,荷兰的邮政编码就是这样的字符串,它包含四位数字和两位字母(如1234AB)。

另外还需要特别提到的是“Replace in string (字符串替换)”步骤。看上去替换字符串很简单,它只是用某个字符串来替换字符串里的部分数据,但是这个步骤可以使用正则表达式,这样这个步骤的功能就会非常强大和灵活。在Kettle自带的例子里有一个“字符串替换”转换,这个转换演示了“字符串替换”步骤里所有可用的参数。其他有用的步骤还包括“splitting fields (拆分字段)”和“splitting fields to rows (拆分字段成行)”,这两个步骤使用分隔符拆分字段,以及“String cut (字符串剪切)”步骤、“Value Mapper (值映射)”步骤等。值映射步骤是使用一个标准的值替换字段里的其他值。例如第4章的load_dim_staff转换使用了一个值映射步骤把“Y”和“N”分别替换成“YES”和“NO”。

使用上面描述的步骤,可以完成大部分的数据清洗工作。但也有一些工作要通过写代码完成,将在本章后面介绍。另外还有一些实现特定功能的校验步骤可以用来做数据清洗工作,如信用卡号码校验、电子邮箱校验等,我们也在后面介绍。

这里还要特别提到一个查询步骤,这个查询步骤也是基于规则的查询,但它不返回True/False,它使用了模糊匹配的算法。这就是在“数据排重”部分要介绍的“Fuzzy match (模糊匹

配)”步骤。这个步骤里包含了很多字符串匹配算法。这些算法在计算器步骤中也有一些，下面介绍一下这几个算法的功能和区别。

Kettle 里的字符串匹配算法

Kettle里有两个地方能找到相似度算法：“Calculator”步骤和“Fuzzy match”步骤。这两个步骤里的相似度算法几乎一样，但是工作方式不一样；“Calculator”步骤比较一行里的两个字段，而“Fuzzy match”步骤使用查询的方式，从字典表中查询出相似度在一定范围内的记录。我们在使用这些步骤和算法之前，最好能理解这些算法的概念和用途。打开“Fuzzy match”步骤，选中算法列表，可以看到有很多可用的算法，如“Damerau-Levenshtein”、“Jaro Winkler”、“Double Metaphone”。这些算法的共同点就是用来做字符串匹配。但是它们的工作方式不一样，所以不同算法适合做不同的工作，如排重之类的工作。下面解释一下这些算法。

- **Levenshtein和Damerau-Levenshtein**：编辑一个字符串到另一个字符串需要多少步骤，根据步骤数，来计算两个字符串之间的距离。第一个算法里的编辑步骤只包括插入字符、删除字符、更新字符，第二个算法里还包括调换字符位置的步骤。需要的最少步骤数就是最后的结果，例如，“CASTERS”和“CASTRO”这两个字符串的距离就是2（步骤1：删除字母E；步骤2：把字母S替换为O）。
- **Needleman-Wunsch**：这个算法主要用于生物信息学领域，以差异扣分的方法来计算距离，上面的CASTERS和CASTRO的距离是-2。
- **Jaro和Jaro-Winkler**：计算两个字符串的相似度，结果是介于0（完全不一样）到1（完全一样）之间的小数。例如我们使用Levenshtein 算法计算CASTERS和POOH相似度时，结果是7，而使用Jaro或Jaro-Winkler 算法时，结果是0，因为这两个字符串之间没有任何相似度。而Levenshtein 算法总会得到一个距离，因为一个字符串经过编辑总是可以变成另一个字符串。
- **Pair Letters similarity**：只有“Fuzzy match”步骤里有这个算法，这个算法把两个字符串都分割成多个字符对，然后比较这些字符对。如CASTERS和CASTRO这个例子，这两个字符串被分割为{CA, AS, ST, TE, ER, RS}和{CA, AS, ST, TR, RO}。最后的相似度=（相等的字符对个数）乘以2/两个字符串的字符对总个数，这个例子就是 $(3*2)/11=0.545$ （这个结果是匹配度非常高的）。
- **Metaphone、Double Metaphone、Soundex和RefinedSoundEx**：这些算法都是利用单词的发音来做匹配，也称为语音算法。这些语音算法的缺点是以英语为基础，所以基本不能用于法语、西班牙语、荷兰语等其他语种。

当然，最后的问题就是我们应该选择哪个算法，这取决于你要解决的问题。对信息提取类的需求，例如，提取亚马逊上所有的图书，找出和Pentaho相关的图书，并给出相似度分数，这时就可以使用字符对的模糊匹配算法。如果要做数据排重工作，而且还可能有拼写错误，这时就要使用Jaro-Winkler 算法。但使用该算法时要小心匹配不准确的情况，如“Jos van Dongen”和“Davidson”这两个人名的相似度是0.615，而“Jos van Dongen”和“Dongen, J van”这两个人名的相似度是0.604，从计算结果看前面一对人名相似度更高，但我们明显可以看出后面的一对人名实际是同一个人。

7.1.2 使用参照表

在很多场合下，如更正地址信息等，需要访问外部的主数据或参照数据（或字典表）。基础数据类的应用一般都使用这些数据，如显示出标准的国家名和州名。但很多其他类的应用，数据并不规范。例如，你有一个客服系统，只能通过文本输入的方式输入产品信息。为了清理这些数据，就需要把文本方式输入的产品名称和主数据里的产品名称进行匹配，主数据里的产品名称是最完整和标准的。最常用的使用主数据的场景就是做地址更正。如果地址主数据是免费的，那你很幸运；通常这些地址主数据都是使用订阅方式才能得到的，有一些公司维护和销售这些地址主数据。

提示：www.geonames.org这个网站提供了很多地址方面的Web服务，如国家、城市、邮政编码的查询和校验，甚至还提供了美国的地址级别的信息。

使用查询表使数据一致

参照表有很多用途，一个最常见的用途就是做数据的查询和检验。提供一个输入字段，如果输入字段里的值没有匹配上，就给对应的数据行做一个错误标志。因为只需要找出错误的数据，所以返回的数据都是有限的错误数据。为了提高成功查询的比例，在做查询之前，最好先使用其他步骤清洗数据。

我们使用城市和邮政编码查询做个例子。几乎所有国家都有邮政编码体系，邮政编码连接到区域、城市或特定的街道（如荷兰）。德国的邮政编码体系是世界上最精准的，通过邮政编码和门牌号就能定位到一个唯一的地址。但人们输入邮政编码时，有时会犯错误，所以要检验邮政编码和地址是否匹配。下面的例子演示了如何使用计算器步骤和查询步骤来判断地址和邮政编码是否匹配。

首先，需要一些输入数据，这个例子我们使用了“Data Grid”步骤，并添加一些数据，如图7-1所示。

第一个清洗步骤就是从邮政编码里提取数字，要使用计算器步骤。在计算器步骤里选择“Return only digits from string A”，新增加一个字段保存这些数字，字段名使用像PC4这样有业务含义的字段名。然后就需要找一个参照表。我们也可以使用“Data Grid”步骤来模拟一个参照表，表如图7-2所示。



图7-1 要清洗的数据

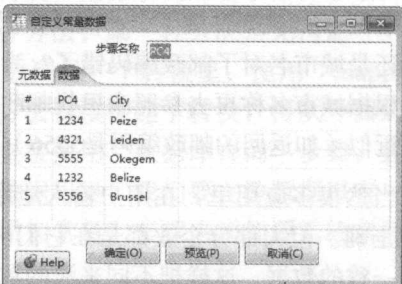


图7-2 参照数据

我们根据PC4字段里的四位数字，再使用“Stream lookup（流查询）”步骤从参照表中查询城市名称。在这个例子里，每次查询都会成功，但是在现实场景中，可能要查询的数据在参照表中没有。为了后面再处理这些没有查询到的数据，建议在查询失败时，使用一个

容易识别的默认值。图7-3显示了完整的流查询步骤，这里我们设置的查询失败的默认值是“***unknown***”。

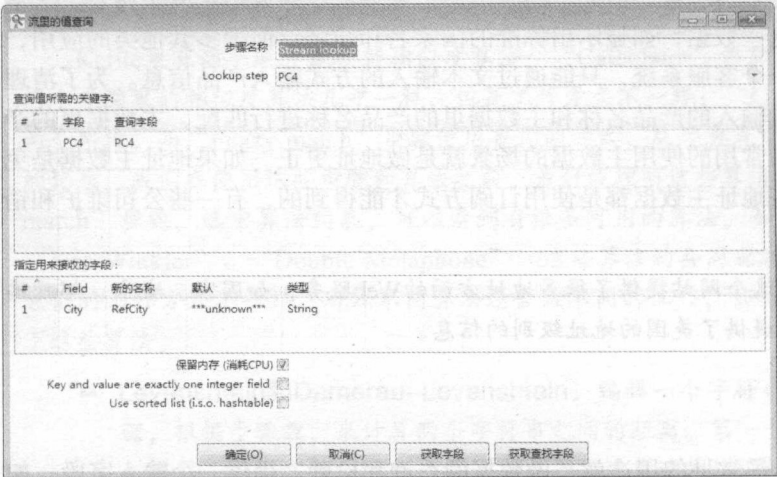


图7-3 流查询

这里我们设置的默认值的前缀和后缀都是***，这样设置有两个目的：首先，检查数据的时候比较容易找到这些异常数据；其次，查询后在模糊匹配原始输入的城市名时，这个默认值不会和原来的任何城市名有相似度。后面模糊匹配的目的主要是为了检查一些拼写或完整性错误。再使用另一个计算器步骤，把City和RefCity作为字段A和字段B，使用 Jaro-Winkler 匹配算法，把新生成的字段命名为cityscore。此时预览数据，可以看到如图7-4所示的结果。



图7-4 预览原始数据和参照数据

我们看第一条记录原始数据和参照数据的相似度非常高，所以这里可能有一些拼写错误。第二条记录是完全正确的。第三条记录看上去有一些问题：城市名称完全不同，另外相似度非常低（一般低于0.8就要怀疑了）。但从数据里我们不能判断出错误在哪里：是邮政编码对了城市名错了？还是城市名对了邮政编码错了？为了得到结论，我们还要做一次相反的校验，“相反”校验是指根据城市名称再去参照表里找邮政编码，然后再和原始数据的邮政编码比较，如果邮政编码非常近似（如返回的邮政编码是5556），我们就可以得出结论，是邮政编码拼写错误。

当然很多应用里，在用户输入地址、邮政编码和城市名时，都会自动检查它们之间的组合是否正确。但大部分企业都是在它们内部的应用里做这些检查，而面向客户的网站一般允许很多不一致的数据，这样把不同来源的数据整合到一起就比较困难了。

使用参照表使数据一致

有一种参照表叫数据确认主表。性别编码就是这种参照表的例子。有的系统使用字母M、F和U，分别代码男、女、未知；有的系统使用 NULL来代表未知的性别；有的系统使用Male和

Female代表男，女；而有的系统则使用完全不同的编码，如0（男）、1（女）或0（未知），1（男）、2（女），等等。还有更复杂的情况，有的系统使用C代表儿童，使用F代表父亲，M代表母亲，各种变化和组合都有可能。要把从这些来源的数据整合到一起，要有一套统一的编码规范，然后把已有的编码映射到规范的编码上。使用单一的查询表比每个系统都有一个查询表要更好，便于维护。

这里要满足两个基本的需求：

- 源系统中的每个可能的值都需要映射。
- 要映射到唯一的一组值。

基于我们前面说的性别的例子，我们需要有表7-1这样的主表。ref_code和ref_name字段，是我们要获取的标准数据，src_system字段是你数据来源于哪个应用或系统，src_code字段包含了这个系统里可能的值。

表7-1 性别编码参照表

| ID | REF_CODE | REF_NAME | SRC_SYSTEM | SRC_CODE |
|----|----------|----------|------------|----------|
| 1 | M | Male | Sales | 1 |
| 2 | F | Female | Sales | 2 |
| 3 | M | Male | Web | male |
| 4 | F | Female | Web | female |
| 5 | M | Male | CRM | F |
| 6 | F | Female | CRM | M |
| 7 | U | Unknown | CRM | C |

这只是一个例子，为了便于查询，数据是以非正规化的结构来组织的。但这种结构适合我们的要求，而且容易查询，根据源系统的名称和原始的数据，就能查询到标准的三个值：M、F、U。

我们还使用 sakila_dwh 库来演示上面的例子，了解工作过程。

说明：因为sakila数据库没有性别编码，我们使用www.fakenamegenerator.com提供的一个客户数据文件，作为下面例子的输入。

像下面这样校验数据的例子应该做成可以重用的子转换，把源系统的名字作为参数传给子转换，子转换就可以设计成通用的了。设计子转换有两种方法，第一种方法是先构建一个完整的转换，然后把完整转换里的性别查询步骤剪切出去，放到另一个转换里，使用“mapping”步骤代替剪切出去的步骤，并设置好传入参数。第二种方法直接构建子转换，再从外面的转换里调用这个子转换。无论哪种方式，你都发现我们不能直接使用“数据库查询”步骤，数据库查询步骤直接基于数据行来过滤，但源系统的名称不在数据行里，我们要在源数据里有一列说明，说明数据来源于哪个源系统。你可能会想到使用“增加常量”步骤来增加源系统名列，如图7-5所示。

但只为了一个过滤条件，就增加一个字段的这种方式好像不太好。而且数据库查询的缓存选项会把所有的数据缓存到内存里。我们的参照表例子只有几行，这样做可以。但在实际生产环境中，如果你是从一个很大的参照表里只查找几个数据，把参照表都加载到内存里就会消耗资源。

一个更好的方式是使用变量来过滤数据，另外使用表输入和流查询步骤。图7-6显示了转换流程的主要结构。

| on | Domain | system_constant |
|---------------------|------------------------|-----------------|
| rmation officer | SeriousBlog.com | Web |
| ar | FuelPlants.com | Web |
| spairer | URLRankings.com | Web |
| | AwesomePoetry.com | Web |
| otographer | AbacusMath.com | Web |
| machine tool setter | ChestPimples.com | Web |
| controller | Airzilla.com | Web |
| | AutomobileRetailer.com | Web |
| irector | TacomaHandyman.com | Web |
| ods operator | BuyCherries.com | Web |

图7-5 把源系统名作为一个常量

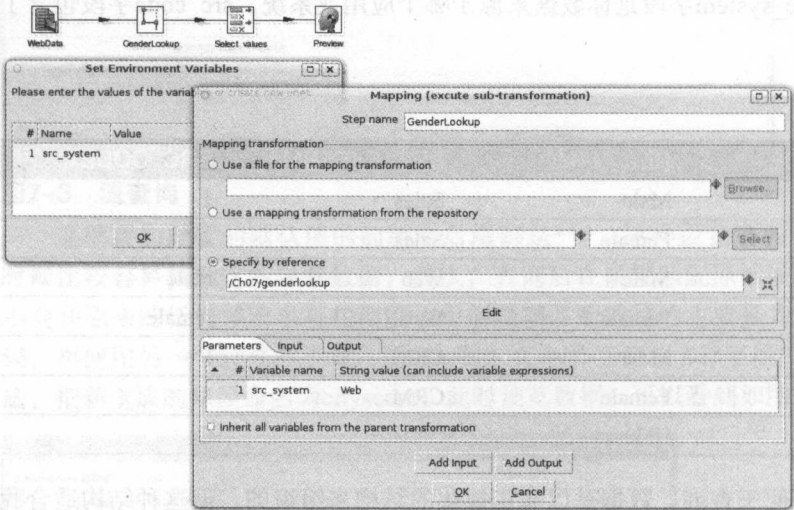


图7-6 传递一个变量

首先，要给予转换步骤定义一个变量，在映射步骤的“参数”标签下设置变量。在这个例子里，我们把值为Web的变量传递给转换genderlookup。图7-7显示了如何在子转换里使用这个变量。

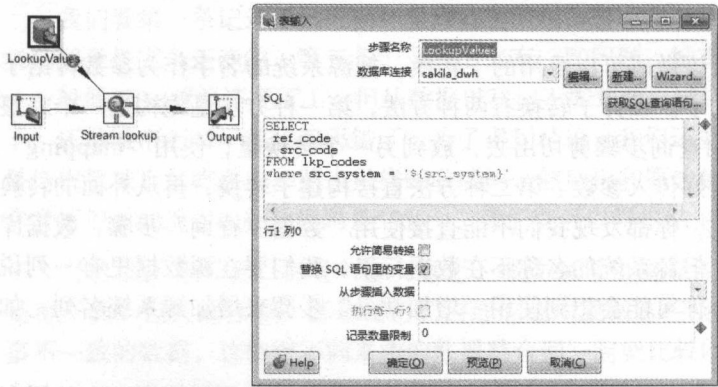


图7-7 使用传来的变量

转换里的流查询步骤非常简单；只需设置好条件，输入数据的src_code字段等于参照表的src_code字段，并指定要返回的字段即可。注意这里必须要设置一个默认值，来处理NULL和未

知的值。完整的流查询步骤如图7-8所示。

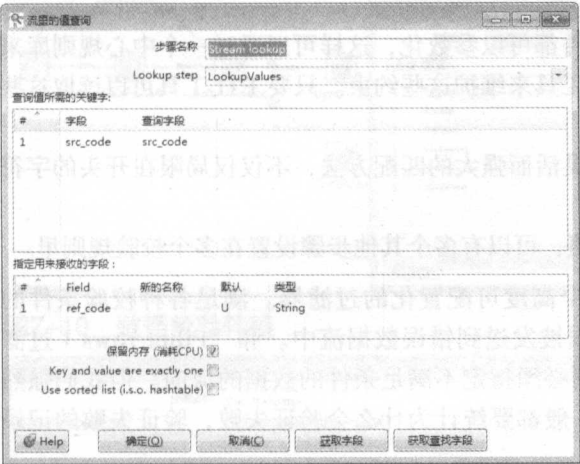


图7-8 “Stream lookup (流查询)” 步骤

说明： 原始数据里可能会包含NULL值，但NULL并不是数据库里的一个真正的值。NULL=NULL这样的比较条件也不会成功。这就是为什么我们没有把对 NULL 值的匹配放到参照表里，以及为什么在流查询步骤里一定要设置一个默认值的原因。

7.1.3 数据校验

数据一般都要遵守一定的规则。在我们前面参照表的例子里，数据必须是参照表里定义的格式。基于参照表的数据验证比较简单，因为符合规则的数据都已经在参照表中事先定义好了。但是大多数数据校验比基于参数表的这种方式更复杂，例如下面的几种情况：

- 电子邮箱的地址必须是有效的格式。
- 输入的数据都必须是大写/小写。
- 日期必须是dd-mm-yyyy的格式。
- 电话号码必须是xxxx-xxxx-xxxx的格式。
- 数值不能超过最大值X。
- 订阅用户必须大于18岁。
- IBAN（国际银行账户号）必须是有效的。

我们还能列出上百个和上面类似的例子，但这些例子都有一个共同点：检查数据是否遵照预定义的业务规则，要找出不符合业务规则的数据。注意，这些都属于“属性值域约束”的范畴。在Kettle 里能完成这些校验功能的就是“Data Validator（数据校验）”步骤。第一次编辑“数据校验”步骤的时候，这个步骤是空的。可以选择“增加检验”来创建一个校验，创建校验时可以看到有很多选项，其实没有这么复杂，我们根据不同的数据类型只要设置其中的部分选项就可以了。我们先看看这个步骤的几个特点。

- 可以给一个列设置多个约束：有的列可能需要多个约束，这个步骤没有限制在同一个列上可以设置几个约束。
- 校验数据类型：当输入的是字符串格式的数值和日期类型的数据时非常有用，通过设置格式掩码可以很容易地找到非法数据。

- **错误合并**：把这一行里发现的所有错误，连接合并成一个字符分割的字符串，保存到错误描述字段里。
- **约束条件参数化**：几乎所有的约束条件都可以参数化，这样可以通过一个中心规则库来管理这些约束条件，可以不通过 ETL工具来维护这些约束。只要 ETL工具可以读取这些规则并传递给数据验证步骤。
- **正则表达式匹配**：正则表达式是非常灵活而强大的匹配方法，不仅仅局限在开头的字符串、结尾的字符串等简单匹配。
- **查询值**：可以从其他步骤读到允许的值，可以有多个其他步骤设置在多个校验规则里。

和过滤步骤相比，数据校验步骤更像一个高度可配置化的过滤器。满足各种校验条件的数据被发送到主数据流里，不满足条件的数据被发送到错误数据流中。和“Filter rows（过滤行）”步骤不同的是，“数据校验”步骤不用必须指定不满足条件的数据的流向。但我们强烈建议要设置不满足条件的数据的流向。因为一般都要统计为什么会验证失败，验证失败的记录有多少，等等。

应用检验规则

为了解释“数据校验”步骤能做什么，我们创建了一个小数据集的例子，如图7-9所示。

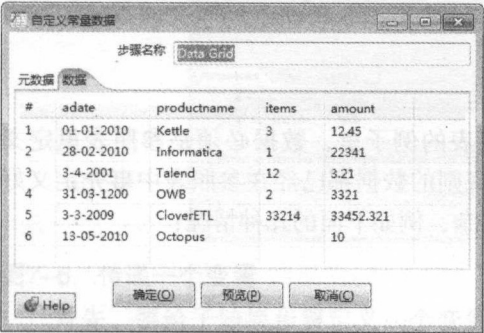


图7-9 要校验的数据

这些数据没有满足下面的规则：

- 字段值都不能为NULL 值。
- 日期不能在2000年1月1日前。
- 产品名称必须是官方提供的名称。
- 产品数量字段必须在1和10之间。
- 单个产品价格不能超过1000元。

最后的一个验证条件稍微麻烦一些，它需要计算，它是“数据校验”步骤唯一不能直接检验的规则。在“数据校验”步骤里不能定义计算字段或派生字段，所以只能在这个步骤前面通过计算器步骤把要检验的数据准备好。图7-10显示了这个数据校验的转换的流程。

现在就可以设置检验规则了。第一个规则是不允许NULL值的规则，我们需要对每个输入的字段都创建一个检验条件。在新建检验的窗口里，有一个“允许NULL”的选项，不选择这个选项即可。第二个规则是针对日期字段的规则，可以在日期字段已有的检验条件上修改，但这会限制后面的错误分析步骤。所以我们最好还是使用一个独立的检验条件，这样，错误数据容易分离出来。创建完每个检验条件后，设置窗口应该如图7-11所示。

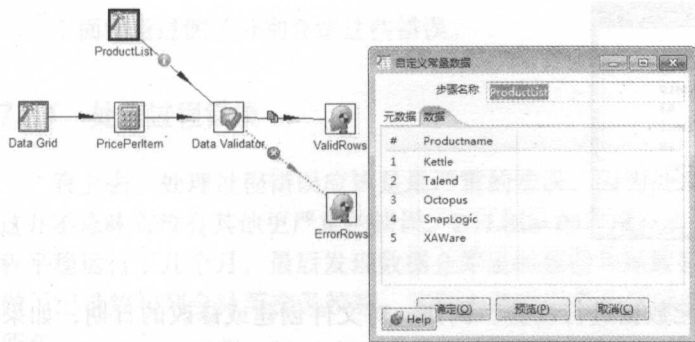


图7-10 数据校验转换

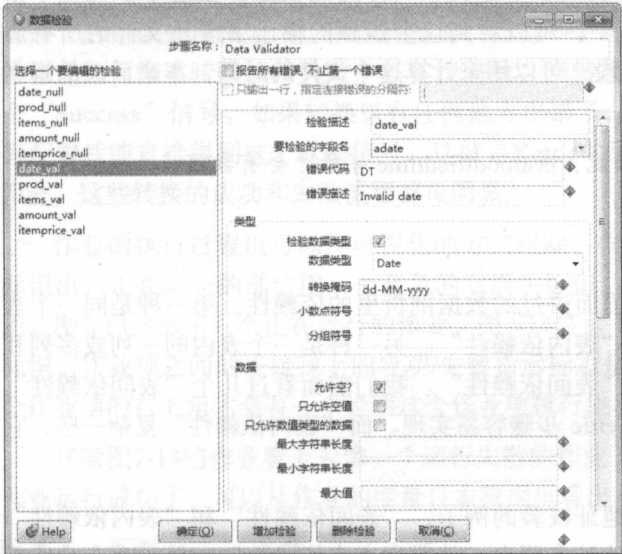


图7-11 数据校验的日期规则

图7-11 显示了一组校验条件。左侧是校验条件的名称，右侧是校验条件的内容。

警告：设置日期的校验条件时，不但要校验数据类型，还要设置掩码（日期格式）。如果没有指定日期格式，就会使用默认的系统日期格式。如果你默认的系统日期格式是yyyy/mm/dd而输入数据的日期格式是dd-mm-yyyy，Kettle 就会抛出异常。

第二个校验条件，日期不能在2000年1月1日前，已经设置完了。第三个校验条件也很简单，选中“从其他步骤获得允许的值”并选择“ProductList”步骤和这个步骤下的“Product”字段作为输入字段。“ProductList”步骤和“数据校验”步骤之间的连线可以手工创建，也可以通过“数据校验”步骤的浮动菜单的“左箭头”按钮创建，如果使用第二种方式，单击左箭头按钮后，再把鼠标移动到“ProductList”步骤并单击，就会出现一个菜单，里面有 optional reference data for Validation <validation_name> 等选项。这些选项可以在已有的检验条件里把“ProductList”步骤设置为参照步骤，也可以新建一个检验条件，并把“ProductList”步骤作为参照步骤。这个连线上有个带圆圈的字母 i，表示这个连线是一个信息数据流，如图7-10所示。

第四个规则，数量必须在1和10之间，也容易设置：把1设置为最小值，把10设置为最大值就可以了。同样最后一个规则，把1000设置为最大值即可。现在我们可以可以在“ValidRows”和“ErrorRows”步骤上预览数据，来检查数据校验步骤是否生效，结果如图7-12 所示。

| # | adate | productname | items | amount | ItemPrice |
|---|------------|-------------|--------|-----------|-----------|
| 1 | 28-02-2010 | Informatica | 1 | 22341.0 | 22341.0 |
| 2 | 03-04-2001 | Talend | 12 | 3.21 | 0.3 |
| 3 | 31-03-1200 | OWB | 2 | 3321.0 | 1660.5 |
| 4 | 03-03-2009 | CloverETL | 33214 | 33452.321 | 1.0 |
| 5 | 13-05-2010 | Octopus | <null> | 10.0 | <null> |

图7-12 数据校验结果

不仅限于数据，我们还可以对元数据进行检验。例如，对文件创建或修改的日期，如果日期是10天前的日期，转换就停止。为了构建这样的转换，我们可以使用“获取文件名”步骤，这个步骤除了获取文件名，还能获取很多其他关于文件的元数据。在这个步骤上单击右键，在右键菜单里选择“显示输出字段”，可以看到这个步骤的输出字段是文件的所有属性。其中一个属性就是 lastmodifiedtime 字段，可以用来计算这个文件的时间，本章后面的脚本（Scripting）步骤描述了如何计算时间。

说明：只有底层的文件系统支持这个功能，lastmodifiedtime 字段才会有数据。

校验依赖性约束

有两种依赖性约束，非常类似于我们前面讲过的数据剖析里的依赖性。第一种是同一个表内，两列或多列之间的依赖关系，也称为“表内依赖性”。另一种是一个表内的一列或多列和其他表的一列或多列有依赖关系，也称为“表间依赖性”。我们前面看过几个“表间依赖性”的例子，基本上就是查询校验问题，使用Kettle 步骤容易实现。而“表内依赖性”复杂一些，它通常需要在转换里做一些额外的准备工作。

本章后面的数据排重部分会演示一个地址校验的例子，“表间依赖性”和“表内依赖性”在这个例子里同时存在，至少使用了外部地址参照表。关于“表内依赖性”，另一个有难度的例子是名字和性别的依赖；如果数据里既有名字也有性别，就有可能使用名字去检验性别或者相反。但很多名字既可以用于男性也可以用于女性，所以这种依赖检测会有一定误差。

7.2 错误处理

错误处理的目的很明确：你希望你的ETL作业或转换可以捕获并处理运行过程中发生的任何错误。但错误的种类是不同的，如下所示。

- **处理过程错误：**因为技术原因导致处理过程不能继续的错误。可能是文件没找到，可能是服务器关了（或宕机），或服务器密码变化了而没有通知ETL团队，等等。
- **数据校验错误：**一些数据不能通过数据校验步骤。根据错误的影响，转换可以继续执行，这种错误不是处理过程错误。
- **过滤器错误：**实际这不是错误，只不过“过滤行”步骤需要两个输出步骤，一个接收通过过滤器的数据，一个接收没有通过过滤器的数据。经常使用“空操作”步骤接收后面这一类数据。
- **一般步骤错误：**Kettle 里的很多步骤都可以定义错误处理步骤，错误处理步骤用来接收当前步骤处理错误的的数据。

下面将通过例子分别介绍这些错误。

7.2.1 处理过程错误

看上去，处理过程错误应该是最严重的错误，因为处理过程错误会让转换完全停下来。但这并不意味着没有其他更严重的错误。ETL团队的梦魇其实是那种没有错误的错误：一个ETL过程平稳运行了几个月，最后发现数据仓库里的数据和源数据库里的数据不一致。如果在ETL里做了记录数和列合计等交叉校验，实际上就可以避免这种情况的发生。这也就是校验的重要性所在。

在第4章的load_rentals 作业里，我们使用了过程错误处理。作业的每个转换都有一个“发送邮件”的错误处理作业项。但在转换里，没有一个特定步骤可以让转换失败并触发错误。外面的作业如何知道转换执行失败？其实很简单。当转换里所有的步骤都运行成功，转换会返回一个“Success”信号。如果转换里有任何地方出错了，会返回一个“Failure”信号。你不用做任何配置就能直接得到这个返回信号，这就是Kettle的设计方式。图7-15显示了一个有多个转换的作业，这些转换的成功和失败流程都很明显。

作业的执行过程也可以用可视化的方式跟踪，作业里成功执行的部分都使用绿色的对钩标识出，正在运行的部分用一个蓝色的双箭头标识，错误执行部分使用一个红色的停止符号标识。图7-13显示了一个正在运行的作业，图7-14显示了发生了错误的作业。注意两个地方有指示标识，作业项之间的连接线上的成功/失败指示标志是一直都可以看到的，作业开始运行后，每个作业项的右上角才会有一个说明这个作业项执行成功/失败的指示标志。

尽管图7-14的作业看上去像一个运行失败的作业，但事实上这个作业并没有运行失败。这个作业运行成功了，可以从作业的度量日志数据里看出，如图7-15所示。



图7-13 一个正在运行的作业

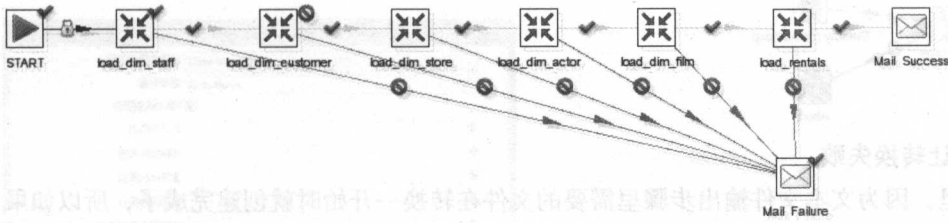


图7-14 一个失败的作业

| | | | |
|-------------------|------------------------|---------|-----------------------------|
| load_dim_staff | Job execution finished | Success | |
| load_dim_customer | Start of job execution | | Followed link after success |
| load_dim_customer | Job execution finished | Failure | |
| Mail Failure | Start of job execution | | Followed link after failure |
| Mail Failure | Job execution finished | Success | |
| job: load_rentals | Job execution finished | Success | finished |

图7-15 失败作业的日志

只有当“Mail Failure”作业项也执行失败了，这个作业才是运行失败。这告诉我们，作业最

后执行的那个作业项才决定着一个作业是否执行成功。在这个例子里, “Mail Failure” 作业项是最后执行的作业项, 如果它执行成功了, 整个作业就执行成功了。如果这并不是主作业, 而是子作业, 看上去好像也没什么问题。但为了清楚起见, 最好在发送邮件步骤错了以后, 让这个作业能明确地返回一个错误状态。“Abort job” 作业项可以做这样的事情, 可以直接放在“Mail Failure” 作业项的后面, 如图7-16所示。

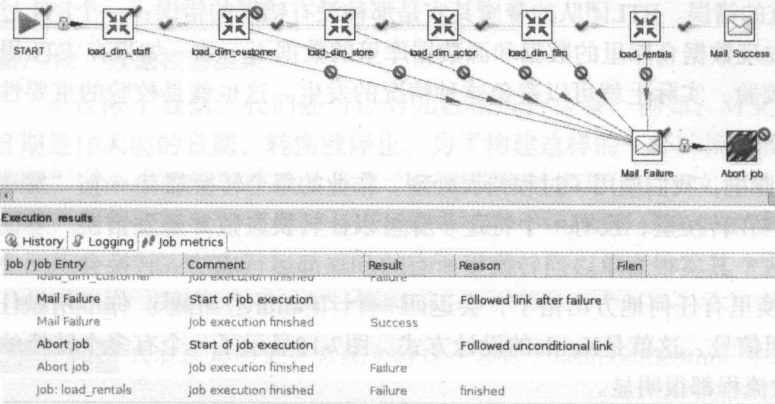


图7-16 强行返回失败结果

这里要注意的是, 因为发送邮件步骤也可能失败, 所以从“Mail Failure” 作业项到“Abort job” 作业项我们使用的是一个无条件的连接。在这个特例里, 尽管作业失败了但可能输出的结果是正确的, 但其他场合并不一定会这样。

7.2.2 转换错误

因为作业可以按照顺序执行里面的作业项, 所以当发生错误时, 可以很容易在一个确定位置终止作业。但转换不能这样, 因为转换里所有的步骤都是同时启动的。如果你希望在不满足某个条件时, 让转换停止运行, 想想应该怎么做? 图7-17是大多数 Kettle 初学者的设计结果。

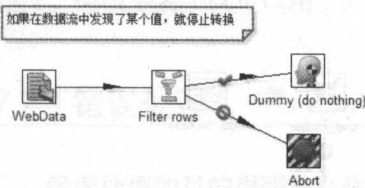


图7-17 强制让转换失败

在图7-17里, 因为文本文件输出步骤里需要的文件在转换一开始时就创建完成了, 所以如果转换里的数据量较少或者第一条记录就不满足条件, 这个转换可以被中止。但在实际情况中, 数据源里有一定量的数据, 在中止转换之前, 已经有一部分数据写到了文本文件中, 实际上在发现不满足条件的记录之前, 这些数据不能被写到文本文件中。数据验证也有同样的问题, 只有满足了所有的验证条件, 数据才能做后续的处理。实际要完成这个工作, 我们需要两个转换, 如图7-18所示。

图7-18里的作业做了两件事情: 第一个是验证 (右下角的图), 验证完之后返回一个成功信号, 才开始后面的转换。如果验证转换运行失败, 作业停止运行, 不会有任何数据被处理。我们也可以对数据库、文件、表、列等做类似的检查, 作业里也提供了这些检查功能。

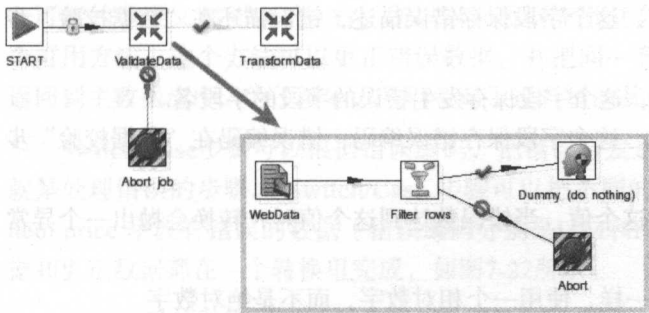


图7-18 中止一个作业

7.2.3 处理数据（校验）错误

我们前面介绍过数据校验步骤，类似一个功能强大的过滤器，但它可以做更多的事情。这个步骤可以告诉你拒绝某一行的准确的原因，这样你可以采取相应的解决办法。我们看一下“数据校验”步骤的前两个复选框（参考前面的图7-11）。

- 报告所有错误，不只第一个：一条记录里可能会不满足多个校验条件。如果选择了这个复选框，即使已经发现一条记录不满足某个验证条件，还是要用其他验证条件来验证这条记录，这样可以获取这条记录的所有错误。
- 输出一行，使用分隔符连接所有错误：当选中这个选项，一行记录的所有错误都放在一行的一个字段里，类似于MySQL的 group_concat 函数。如果选中了“报告所有错误”选项，而没有选中这个选项，所有的错误就会放到多行里。

下面还有两个要注意的选项是“错误编码”和“错误描述”，这两个选项关系到对每个校验规则如何报告错误。如果要在校验失败后，继续处理失败的数据，若使用一组规范的错误编码，会便于后面的错误处理转换。

在开始错误处理工作之前，需要先启用错误处理。定义了错误处理可以把错误的数据行传到另一个步骤里。错误编码和错误描述字段也被增加到错误行里。要启用错误处理，右键单击“数据校验”步骤并选中“定义错误处理”，会打开错误数据处理对话框，如图7-19所示。



图7-19 定义错误处理

在这个对话框里，可以设置如何处理错误数据。这里的一些选项和“数据校验”步骤里的错误处理的选项有关。

- 错误字段名：设置一个字段名，这个字段保存错误的总个数，如果“数据校验”步骤里没有选中“输出一行”，这个字段的值永远是1。

- 错误描述列名: 设置一个字段名, 这个字段保存错误描述, 错误描述在“数据校验”步骤里设置。
- 错误列的列名: 设置一个字段名, 这个字段保存发生错误的字段的字段名。
- 错误编码列名: 设置一个字段名, 这个字段保存错误编码, 错误编码在“数据校验”步骤里设置。
- 允许的最大错误数: 如果设置了这个值, 当错误数达到这个值时, 转换会抛出一个异常(失败代码)。
- 允许的最大错误百分比: 和上面一样, 使用一个相对数字, 而不是绝对数字。
- 在计算百分比前最少要读入的行数: 开始时不计算百分比, 直到读入的数据达到设定的值时才开始计算。如果没有设置这个值, 而设置了最大百分比是10%, 那么在前九条里, 只要有一条错误的的数据, Kettle就会停止转换。

提示: 我们在处理验证错误时, 有下面一些要注意的地方:

- 如果在“数据校验”步骤里没有指定“错误编码”和“错误描述”, Kettle 会自动创建错误编码和描述。错误编码的形式是KV###, ###是数字。错误描述类似于“During validation of field ‘items’ we found that its value is null in row <[13-05-2010], [Octopus], [null], [10.0], [null]> when this is not allowed.”。尽管错误描述的信息比较详细, 使用自己定义的简单描述可能更好。
- 开发人员总喜欢在一个校验里放入多个验证条件, 但这样在检验失败时, 会不清楚是哪个错误。最好把每个检验条件放在不同的检验里。如一个检验是验证某个字段是否为NULL, 另一个检验是验证同一个字段的取值范围。

有了上面这些知识, 你就可以创建数据清洗流程, 来处理没有通过校验的数据。首先, 我们先看“数据校验”的结果。错误数据的结果如图7-20所示。

我们可以看到, 数据里有很多错误。看最后的两行: items和itemPrice都有NULL值。我们在设置“数据校验”时, 在已有的数据校验里, 都选中了“允许NULL”复选框。而把NULL值验证放在了另外独立的校验规则里, 我们才能看到上面的结果。如果我们不这么做, 而是创建一个校验来检查多个条件, 我们就会得到如图7-21所示的结果, 这个结果看上去就没有上一个结果明确。

| # | adate | name | items | amount | ItemPrice | error_desc | error_field | error_code |
|---|------------|-------------|-------|-----------|-----------|---------------------------|-------------|------------|
| 1 | 28-02-2010 | Informatica | 1 | 22341.0 | 22341.0 | Invalid Product | name | PN |
| 2 | 28-02-2010 | Informatica | 1 | 22341.0 | 22341.0 | Too expensive! | ItemPrice | PR |
| 3 | 03-04-2001 | Talend | 12 | 3.21 | 0.3 | Nr of items outside range | items | ITM |
| 4 | 31-03-1200 | OWB | 2 | 3321.0 | 1660.5 | Wrong date | adate | DT |
| 5 | 31-03-1200 | OWB | 2 | 3321.0 | 1660.5 | Invalid Product | name | PN |
| 6 | 31-03-1200 | OWB | 2 | 3321.0 | 1660.5 | Too expensive! | ItemPrice | PR |
| 7 | 03-03-2009 | CloverETL | 33214 | 33452.321 | 1.0 | Nr of items outside range | items | ITM |
| 8 | 13-05-2010 | Octopus | | 10.0 | | items has NULL value | items | NULL |
| 9 | 13-05-2010 | Octopus | | 10.0 | | Item price has NULL value | ItemPrice | NULL |

图7-20 校验错误

| # | adate | name | items | amount | ItemPrice | error_desc | error_field | error_code |
|---|------------|-----------|-------|-----------|-----------|---------------------------|-------------|------------|
| 7 | 03-03-2009 | CloverETL | 33214 | 33452.321 | 1.0 | Nr of items outside range | items | ITM |
| 8 | 13-05-2010 | Octopus | | 10.0 | | Nr of items outside range | items | ITM |
| 9 | 13-05-2010 | Octopus | | 10.0 | | Too expensive! | ItemPrice | PR |

图7-21 混乱的错误描述

图7-20 演示了唯一而简洁的错误编码的重要性: 根据这些错误编码, 后面就可以使用 Switch/Case 步骤把这些错误发送到不同的处理流程。这里要注意一点, 因为我们给每行数据每

个可能的错误又都创建了一行，所以错误数据流里就会有重复的数据行。下面的例子演示了一个可用方法，这个方法可以更正错误数据，并把同一数据项的不同错误行合并成一行数据，并返回到主数据流中。为了演示，我们忽略了这行里的其他错误。

Switch/Case步骤可以根据错误编码，把错误行发送到不同数据流中。Switch/Case步骤后面就是处理错误的步骤。Switch/Case 步骤可以把不同的数据发送到相同的输出步骤里，items和 item price 字段有错误的数据（错误编码分别是ITM和PR）都被发送到了同一个步骤里。校验数据和更正数据都在一个转换里完成，如图7-22所示。

图7-22 里的转换说明了如何把“数据校验”步骤的结果发送到 Switch/Case 步骤，然后又如何根据错误编码，发送到其他步骤。在修改完了“adate”和“items”字段后，修改完的错误数据和正确数据又合并到了一起。

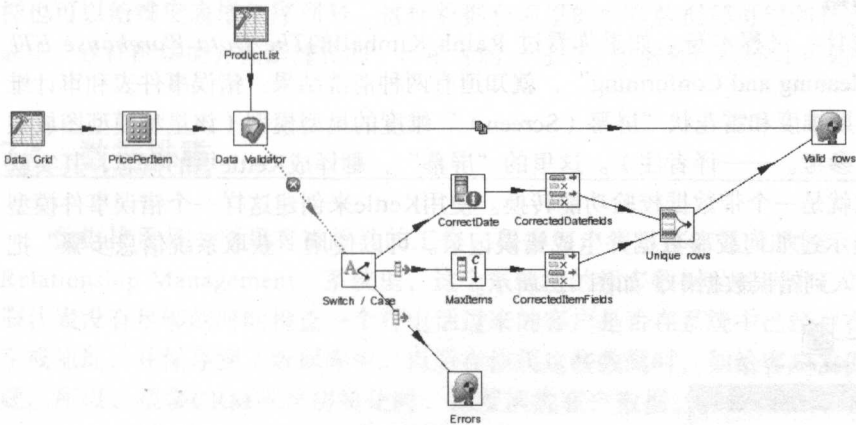


图7-22 校验和更正

注意Switch/Case步骤里只能设置一组固定的值或一些字符串（如果选中了“使用字符串包含比较”）。在 SQL 里，像 CASE WHEN somevalue < 5000 THEN 'Pass' ELSE 'Fail' 这样的语句非常普遍，但在 Switch/Case步骤里不能这样使用。如果你希望Switch/Case步骤可以处理类似上面的条件，我们需要事先把上面的条件翻译成相应的固定值。我们使用“数值范围”步骤可以实现这一功能，如图7-23 所示。

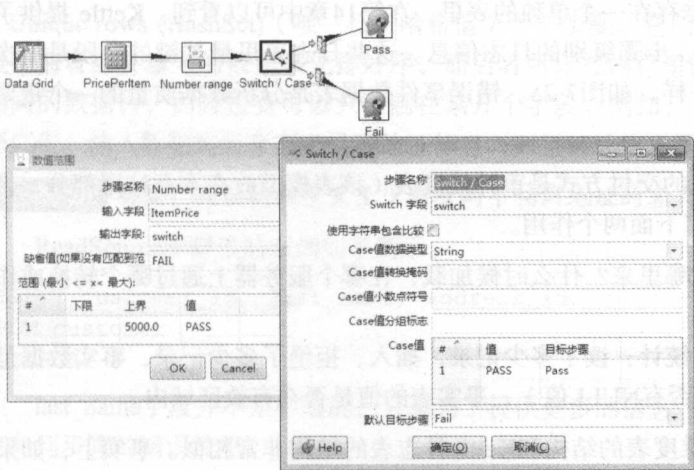


图7-23 准备Case 语句

7.3 审计数据和过程质量

审计转换后数据的质量，通常是提高数据质量过程的第三步。第一步是数据剖析，第二步是检验和错误处理，当最后的结果都被存储在单独的审计或日志里，这些信息就可以用来做报告和分析了。最后一步，要基于审计和验证的结果再更新数据，我们这里不涉及这一步。图7-24显示了“数据质量的生命期”，这是一个永无休止的过程。

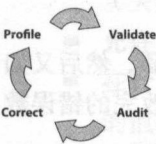


图7-24 数据质量生命期

另外还有第二种审计：过程本身。如果你看过 Ralph Kimball的*The Data Warehouse ETL Toolkit*一书的第4章“Cleaning and Conforming”，就知道有两种清洗结果：错误事件表和审计维度。前者就是一个带日期维度和雪花状“屏幕（Screen）”维度的星型模型（该星型模型图放在本章结尾部分，供读者参考。——译者注）。这里的“屏幕”，翻译成Kettle里的概念，其实就是一个转换，准确地说就是一个带数据校验功能转换。使用Kettle来创建这样一个错误事件模型非常容易。我们已经演示过如何校验数据并生成错误记录。可以使用“获取系统信息步骤”把当前转换批次的信息加入到错误数据中，如图7-25所示。



图7-25 创建错误事件数据

这时可以把这些详细的错误数据存在一个单独的表里。在第14章中可以看到，Kettle 提供了日志和审计功能来存储日志、转换、步骤级别的日志信息。这些日志表里最关键的字段是批次号（batch ID），和错误事件数据一样，如图7-25。错误事件数据表是分析数据质量的一个重要途径。

Kimball 提出的第二种错误数据的交付方式是审计维度表（该表模型放在本章结尾部分，供读者参考。——译者注）。这个表有下面两个作用。

- 提供审计事实细节：数据从哪里来？什么时候加载，在哪个服务器上通过哪个转换或作业？
- 提供基本的数据质量指标和统计：读了多少记录，插入、拒绝了多少记录，事实数据是否完整（即事实表的外键是否有NULL值），事实表的值是否在有效区域内。

细心的读者可能会发现，审计维度表的结构和Kettle日志表的结构非常相似。事实上，如果启用了日志，审计维度表里的大部分内容都可以得到，只要再增加 batch_id（audit key）字段即

可。这个修改如图7-26所示。

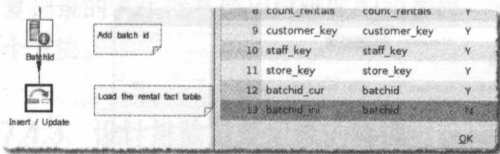


图7-26 增加batch ID 列

我们看到上图中 batch ID 列出现了两次，因为这是一个可更新的事实表，而且原始batch ID 列不应被修改。为了不修改最原始的batch ID 列，而且记录每次修改的batch ID，需要使用两列。一列是静态的（这里的列名是batchid_ini），用来保存插入这条记录时的批次号，另一列是可更新的（这里的列名是batchid_cur）。给事实表增加批次号，不会给事实表造成什么影响。同样也可以给维度表增加序列号。这样数据仓库里的所有数据都可以和日志表、错误事件表关联起来。这样的解决方案是健壮的、可审计的，每一个数据都可以追踪到创建它的过程。

7.4 数据排重

数据排重是一项具有挑战性的工作，很多客户表里都有重复数据，例如CRM（Customer Relationship Management）系统里，这个系统里的很多数据是呼叫中心的客服代表录入的。客服代表没有足够的时间检查一个打电话过来的客户是否在系统中已经存在，或者错误拼写了名字或地址，并保存到了数据库中。以后在使用这些数据时，如给客户发促销邮件，就会发生问题。所以，很多CRM系统初始化时，都要清洗客户数据，保证数据库里一个客户只有一条记录。当然，问题是如何检测出重复的数据？更具有挑战性的是：如何在没有主键的情况下或主键可能拼写错误的情况下，检测到重复数据。遗憾的是，对这类问题，不会有任何一个软件或方法可以百分之百解决。但通过模糊匹配，可以逐渐把数据整理好。

7.4.1 去除完全重复的数据

Kettle有两个去除重复记录的步骤，“Unique rows（去除重复记录）”步骤和与之类似的“Unique rows (HashSet)（唯一行（哈希值））”步骤。它们的工作方式类似而且都易于使用，但是前者需要输入的数据事先排好序，而后者可以在内存里操作。这两个步骤都只能识别完全相同的数据行，而且检查可以只限制在某几个字段。例如，一个公司想给每个客户地址寄送直邮广告。输入数据需要有客户号和地址信息，而且地址信息需要排序。

说明：前提：我们的例子里至少要有两个相同地址的客户信息。

ReadSource步骤里的查询如下所示。

```
SELECT customer_id, last_name, address_id
FROM customer
ORDER BY 3
```

last_name字段并不是必需的，只是为了提供更多的信息。图7-27是转换流程，错误数据被重定向到步骤的错误流里。

“Unique rows”步骤的设置非常简单，如图7-28所示。选中“Redirect Duplicate row”复选

框可以把重复行发送到步骤的错误流中，设置address_id作为判断字段。

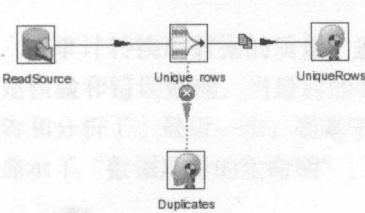


图7-27 去除重复记录

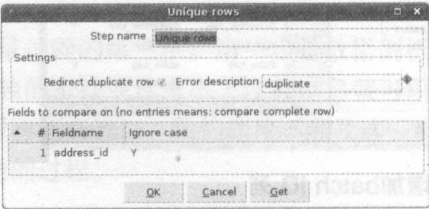


图7-28 “Unique rows” 步骤

两个空操作步骤是为了易于测试，当在“Duplicates”空操作上预览数据，重复数据就会出现，如图7-29所示。

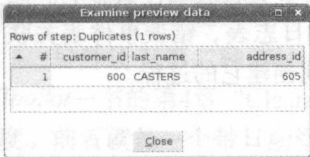


图7-29 步骤预览

7.4.2 不完全重复问题

正如在本节前面介绍的那样，大多数数据质量问题是由客户和产品数据引起的。我们下面的例子都聚焦在客户数据上，因为客户数据易于理解。我们先看一些典型的客户名称重复的例子。假设CRM系统里存储了客户的姓氏、名字、电子邮件地址、城市和国家，如表7-2所示。

可以很容易地看出这两条记录指向同一个人，但如果如果有3条、30条甚至上亿条该怎么办？不能靠眼睛在大量的数据里查找重复的数据。

表7-2 不完全相同的重复数据的例子

| ID | FIRSTNAME | LASTNAME | E-MAIL | CITY | COUNTRY |
|-----|-----------|----------|-------------------------|--------|---------|
| 3 | Rolan | Bouman | Roland.bouman@gmail.com | Leiden | NL |
| 433 | Rolan | Bouman | Rolan.bowman@gmail.com | Leiden | NL |

首先要检查的是一些数据录入点，例如City、Postal code和Country这类字段，这类字段一般是通过选择的方式录入的，系统一般不会让用户自己拼写这些数据，而且系统往往也会检查这些数据的逻辑规则是否正确。无论做何种检查，我们都需要有一些数据正确的列，否则没有信息能把重复的数据关联起来。在表7-2里，城市和国家是正确的数据（至少不会拼写错误）。如果没有这种把可能重复的数据联系到一起的准确数据，排重是根本不可能的。

排重工作下面要做的事情就是要保证有充足的计算能力。检查可能的重复记录意味着要搜索表里的全部记录。如果表里有一百万条记录，检查一条记录就要检查一百万次（1百万×1百万），如果要比较多个字段，情况会更糟。这就是为什么市场上这类工具的价格这么高的其中一个原因。价格高的另一个原因是这类工具都有内置的“知识库”，可以自动地匹配正确的地址和人名信息。

最后，我们还需要算法来匹配可能重复的数据。一些字段的数据是完全一样的，如表7-2所示，一些字段可能拼写错误或拼写方式不同。使用SQL是很难解决这种可能重复数据的问题的：

不能使用相等检测，而“like”检查也不可行，因为要事先指定搜索字符串。唯一能找到可能重复记录的方法就是使用模糊匹配的逻辑，它可以计算字符串的相似度，我们下面就描述一下这个方法。

7.4.3 设计排除重复记录的转换

使用上面提供的信息，不难设计出一个排除重复的方法。我们这里设计的方法包含四个步骤，当然也可以再增加步骤或做一些修改。这些步骤里主要的步骤就是上面提到的“模糊匹配”步骤。这个步骤完成下面的工作：

1. 从数据流里读取输入字段。
2. 使用某一种模糊匹配算法查询另一个数据流里的一个字段。
3. 返回匹配结果。

图7-30显示了这个步骤的一些主要设置。这里要设置一个主数据流里的字段，我们设置“src_lastname”作为主数据流里的字段。另外要设置一个查询数据流和查询数据流里的一个字段。这个步骤的“设置”栏里的选项，主要取决于选择的算法。如果你选择的是“phonetic”算法（或类似的Soundex、Metaphone，等等），没有设置项，“区分大小写”选项也只适用于“Levenshtein”算法。

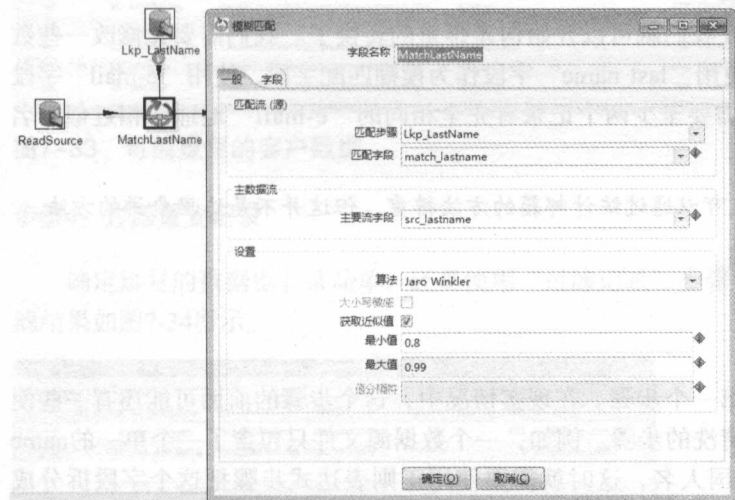


图7-30 模糊匹配基础

其他的一些选项都是用来控制匹配结果的。这里的“获取近似值”选项很重要，这个选项可以只返回一个最相似的数值。如果不选中这个选项，就会返回相似度在最小值和最大值之间的多个数值，多个值使用指定的分隔符分开。图7-31是在没有选中“获取近似值”选项的情况下（使用图7-30的配置），对 sakila 数据库的“last name”列的模糊匹配结果。

在“字段”标签下可以设置匹配列的列名。但因为有多个匹配的数值，以及这些数值也没有可以参照的主键或引用，所以这个结果对后续处理没有太大作用。对于排重工作来说，不但要找到相似的数值，还要知道这些数值属于哪条记录。所以我们最好选中“获取近似值”选项。这个选项只返回相似度最高的数值，另外还返回这个数值所在记录的其他字段，如图7-32所示。在“字段”标签下可以设置查询流里需要的其他字段。

Examine preview data

Rows of step: MatchLastName (600 rows)

| # | src_custid | src_lastname | src_email | match |
|----|------------|--------------|-----------------------------------|---|
| 9 | 9 | MOORE | MARGARET.MOORE@sakilacustomer.org | MORALES.MORRELL |
| 10 | 10 | TAYLOR | DOROTHY.TAYLOR@sakilacustomer.org | |
| 11 | 11 | ANDERSON | LISA.ANDERSON@sakilacustomer.org | SANDERS.GUNDERSON.HENDERSON.ANDREWS.ANDREW.PATTERSON.PEARSON.LARSON.HANSON.NELSON |
| 12 | 12 | THOMAS | NANCY.THOMAS@sakilacustomer.org | THOMPSON |
| 13 | 13 | JACKSON | KAREN.JACKSON@sakilacustomer.org | JACOBS |
| 14 | 14 | WHITE | BETTY.WHITE@sakilacustomer.org | WHITING.WHEAT.HITE |
| 15 | 15 | HARRIS | HELEN.HARRIS@sakilacustomer.org | HARDISON.HARKINS.RICHARDS.HART.ARTIS.HARDER.HARPER.HARRISON |
| 16 | 16 | MARTIN | SANDRA.MARTIN@sakilacustomer.org | MCCARTNEY.MARTEL.ARTIS.MARK.MARTINO.MARTINEZ |

Close

图7-31 预览多个匹配到的数据

模糊匹配

字段名称: MatchLastName

一般 字段

输出字段

匹配字段: match

值字段: score

指定额外的在匹配流中的字段

| # | 字段 | 改名为 |
|---|--------------|-----|
| 1 | match_custid | |

获取字段

确定(O) 取消(C)

图7-32 模糊匹配步骤的返回字段

基于上面学习到的内容，现在我们就可以开始构建排重的转换了。我们需要先修改一些数据。在这个简单的例子里，我们使用“last name”字段作为模糊匹配字段，使用“e-mail”字段作为“参照”字段。因此，我们需要至少两个记录有完全相同的“e-mail”地址和相近似的名字。

说明：在这个例子里，当然也可以通过统计邮箱的方法排重，但这并不是这里要讲的方法。我们下面看转换里的四个步骤。

步骤1: 模糊匹配

在这个例子里，模糊匹配是第一个步骤。在现实情况中，这个步骤的前面可能还有一些使用正则表达式匹配等技术做数据清洗的步骤。例如，一个数据源文件只包含了一个单一的name字段，这是人的全名。如果是美国人名，这时就需要使用正则表达式步骤把这个字段拆分成“first name”、“middle initial”和“last name”字段，如果是其他国家的人名，可能有其他的拆分方法。例如本书其中一个作者的名字，在荷兰是很常见的名字，但名字里包含了“van”，所以美国人经常把作者名字中间的“van”当成了middle name。

模糊匹配步骤和图7-30 里显示的是一样的，数据是通过表输入步骤获取到的，表输入步骤里的SQL如下：

```
SELECT customer_id AS src_custid
,      last_name   AS src_lastname
,      email       AS src_email
FROM   customer
```

除了输入数据，还需要参照数据，用来进行模糊匹配，参照数据来源于同一个表，参照数据也使用了一个表输入步骤来获取，SQL语句如下：

```
SELECT customer_id AS match_custid
,      last_name   AS match_lastname
FROM    customer
```

“模糊匹配”步骤的设置和图7-30相同。

步骤2：选择出疑似记录

我们使用的模糊匹配算法是Jaro-Winkler算法，设置的最小相似度是0.8，不是所有的记录都能找到近似的记录。所以我们要过滤出有相似值的记录，这里要使用“过滤记录”步骤，过滤条件是match_custid IS NOT NULL。符合这一条件的记录被输出到下一个步骤，其他记录被丢弃。

步骤3：查询校验值

这个例子里，我们使用电子邮箱地址作为额外的校验信息，当然我们也可以使用其他的地址字段等。通过模糊匹配后，结果的每条记录里都有两个客户ID列、两个名字列、两个电子邮箱地址列，部分结果如图7-33所示。

Rows of step: Lkp_Email (475 rows)

| # | src_custid | src_lastname | src_email | match | score | match_custid | lkp_email |
|----|------------|--------------|-------------------------------------|-----------|-------|--------------|--------------------------------------|
| 16 | 17 | THOMPSON | DONNA.THOMPSON@sakilacustomer.org | THOMAS | 0.9 | 12 | NANCYTHOMAS@sakilacustomer.org |
| 17 | 18 | GARCIA | CAROL.GARCIA@sakilacustomer.org | GARZA | 0.9 | 224 | PEARL.GARZA@sakilacustomer.org |
| 18 | 19 | MARTINEZ | RUTH.MARTINEZ@sakilacustomer.org | MARTIN | 1 | 16 | SANDRA.MARTIN@sakilacustomer.org |
| 19 | 20 | ROBINSON | SHARON.ROBINSON@sakilacustomer.org | ROBINS. | 1 | 472 | GREG.ROBINS@sakilacustomer.org |
| 20 | 21 | CLARK | MICHELLE.CLARK@sakilacustomer.org | CLARY | 0.9 | 525 | ADRIAN.CLARY@sakilacustomer.org |
| 21 | 22 | RODRIGUEZ | LALITA.RODRIGUEZ@sakilacustomer.org | RODRIGUEZ | 1 | 289 | VIOLET.RODRIGUEZ@sakilacustomer.org |
| 22 | 23 | LEWIS | SARAH.LEWIS@sakilacustomer.org | WILES | 0.9 | 455 | JON.WILES@sakilacustomer.org |
| 23 | 25 | WALKER | DEBORAH.WALKER@sakilacustomer.org | WALTERS | 0.9 | 269 | CASSANDRA.WALTERS@sakilacustomer.org |
| 24 | 26 | HAU | ESSICA.HAU@sakilacustomer.org | HAUS | 0.9 | 386 | PAMONA.HAU@sakilacustomer.org |

图7-33 可能重复的客户数据

步骤4：过滤重复记录

确定重复的数据也非常简单。还是使用“过滤记录”步骤，条件是src_email = lkp_email。过滤结果如图7-34所示。

Examine preview data

Rows of step: View (2 rows)

| # | src_custid | src_lastname | src_email | match | score | match_custid | lkp_email |
|---|------------|--------------|----------------|--------------|-------|--------------|----------------|
| 1 | 1 | van Dongen | joe@tholis.com | van den Dong | 0.9 | 433 | joe@tholis.com |
| 2 | 433 | van den Dong | joe@tholis.com | van Dongen | 0.9 | 1 | joe@tholis.com |

Close

图7-34 可能的重复记录

我们说图7-34里的记录是“可能的”重复记录是有原因的。尽管在这个例子里，可以很肯定说这两个记录是一个人。但实际情况中也有可能是一对夫妻共享了同一个电子邮箱地址。还有更糟的情况，例如，两个记录指向同一个人，但两个地址都是有效的，一个是街道地址，一个是邮局邮箱地址，或者一个人有两个电子邮箱地址，等等。

排重的另一个问题是正确性的问题：即使找到了不同地址或不同电话号码的重复记录，那么哪一个地址或电话号码是正确的？不是所有的业务系统对字段的变化都有详细的变更日志，即使有，还有人为错误输入的可能。

把多个记录合并成一个记录要非常小心。首先，要确定以哪条记录里的数据为准。其次，

地址应该作为一个整体来处理, 如果把一条记录的城市名称, 合并到另一条城市名称为空的记录里通常没有太大意义。基于我们这里提供的一个基本的排重转换, 再结合上面的一些注意问题, 你就可以开始进行排重工作了, 我们这里提供的转换如图7-35所示。

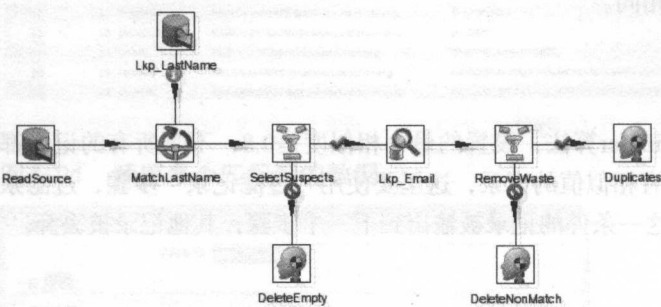


图7-35 排除疑似重复

7.5 脚本

大多数ETL开发人员对脚本都是爱恨交加。我们一方面努力去开发一个不用写代码的易于维护的ETL流程, 而另一方面却不得不为了一些复杂问题, 而必须要通过脚本来实现。在历史上, Java脚本步骤曾经是Kettle的“万能胶”, 有很多复杂的转换都必须使用它来实现。很多年过去了, 出现了越来越多的步骤来实现以前只能靠Java脚本步骤来完成的功能。总的来说, 应该避免使用脚本步骤, 但实际中还会有一些需求, 不能用常规步骤来完成。

在编写本书时, 转换的脚本步骤类别下有七个不同的脚本步骤, 另外还有一个正在开发的脚本步骤。

因为使用SQL脚本的场合较少, 我们下面只讲解几个脚本步骤: 公式、Java脚本、自定义Java表达式、自定义Java类、正则表达式。在深入讲解之前, 下面简要看一下这些脚本步骤的功能。

- **公式 (Formula) :** 实际这不是一个真正的脚本步骤, 但它可以提供比“计算器”步骤更多的灵活的公式。这里使用的公式语法和OpenOffice使用的公式语法相同, 所以如果你熟悉OpenOffice电子表格 Calc里的公式, 你也同样熟悉这里的公式的语法。
- **Java脚本:** 在这个步骤里可以写JavaScript, 可以通过JavaScript读取文件、连接到数据库、输出信息到弹出窗口, 等等。你会发现它能提供比你想要的更多的功能。
- **用户自定义Java表达式:** 可以通过这个步骤直接写Java表达式, 在转换启动后, 这些Java表达式会被编译成Java代码。相对于脚本步骤, 这个步骤的性能更高。
- **用户自定义Java类:** 除了上面单一的Java表达式, 可以通过这个步骤创建一个完整的Java类, 这个步骤可以用来实现一个自定义的Kettle插件, 如何使用这个步骤, 参考Matt Caster的blog, <http://www.ibridge.be/?p=180>。
- **正则表达式:** 使用正则表达式来匹配字符串, 可以使用简单或非常复杂的正则表达式, 能把通过分组 (capture groups) 捕获到的字符串放到新的字段里。

为了实现某个功能而选择步骤, 要在易于使用、快速开发、提高运行效率等因素之间平衡利弊。这些步骤之间的优缺点, 可以参考本书作者的blog, <http://rpbouman.blogspot.com/2009/11/pentaho-dataintegration-javascript.html>。

7.5.1 公式

公式步骤是计算器步骤的近亲，这两个步骤都可以完成很多以前只能通过Java脚本步骤才能完成的工作。当数据要通过某些逻辑的处理，而且计算器步骤里预定义的逻辑不能满足我们的需求时，就可以使用公式步骤。日期运算是一个比较好的例子：通过计算器步骤可以计算两个日期之间相差几天，但不能算出来相差几个月或相差几年。使用公式步骤会解决这个问题，但也会产生一个新问题：只要不在一个月份里的两天，即使这两天之间只间隔了一天，使用datedif函数计算出来的结果也是一个月。

公式步骤里的公式和计算器步骤里的公式也有一些不同的地方。在公式步骤里，不能让步骤本身定义的字段也参与运算。为了让中间产生的计算字段也参与运算，就要使用两个公式步骤。最后，公式步骤里的字段都要手工输入，没有下拉列表的选择，而计算器步骤可以从下拉列表里选择字段。例如，我们本章前面提过的计算文件年龄的功能，下面转换里的前三个步骤的功能分别是“获取文件名”、“增加系统时间”和“选择文件名、系统时间、文件的最后一次更新时间”。

获取文件名的时候，可以获取任意类型的文件名，这个例子里获取的是/etc 目录下的.conf 文件。在三个步骤的后面要创建第一个公式步骤，这个步骤里增加了一个“age_in_months”字段，然后再创建一个公式步骤，用来计算这个文件的年龄（从文件创建到现在的时间）是不是太大了。转换如图7-36所示。

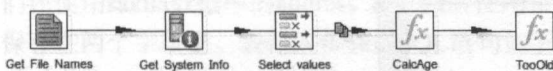


图7-36 使用公式步骤判断文件年龄

转换里的第一个公式是datedif([lastmodifiedtime];[today];"m");，第二个公式是if([age_in_month]>24;"Too Old"; "OK")。输出结果如图7-37所示。

| Examine preview data | | | | | |
|--------------------------------|-----------------------|-------------------------|-------------------------|---------------|---------|
| Rows of step: TooOld (42 rows) | | | | | |
| # | filename | lastmodifiedtime | today | age_in_months | TooOld |
| 9 | /etc/passwd | 2009/03/03 16:42:52.000 | 2010/05/17 00:00:00.000 | 14 | OK |
| 10 | /etc/gai.conf | 2008/03/26 18:44:50.000 | 2010/05/17 00:00:00.000 | 26 | Too Old |
| 11 | /etc/gssapi_mech.conf | 2009/06/08 12:24:44.000 | 2010/05/17 00:00:00.000 | 13 | OK |
| 12 | /etc/ldpam.conf | 2009/10/06 22:36:37.000 | 2010/05/17 00:00:00.000 | 17 | OK |
| 13 | /etc/hosts.conf | 2009/04/20 16:07:21.000 | 2010/05/17 00:00:00.000 | 13 | OK |
| 14 | /etc/hosts.conf | 2008/12/23 19:53:36.000 | 2010/05/17 00:00:00.000 | 31 | Too Old |
| 15 | /etc/ldmapd.conf | 2009/10/20 18:30:31.000 | 2010/05/17 00:00:00.000 | 17 | OK |

图7-37 文件年龄输出结果

7.5.2 Java脚本

随着Kettle 4 版本里增加的大量功能，对Java脚本的需求比以前少了很多。但有时还是需要这个步骤，如第10章里的对数据流的循环。我们这里要讲的例子还是基于sakila数据库的Rentals表。这个表里有租赁和归还的日期，我们想找出租赁间隔（归还时间减租赁时间）超过一个星期的客户。我们首先想到的是计算器步骤或公式步骤，但这两个步骤都不能计算两天之间的星期数。而使用Java脚本步骤可以容易计算出星期数。首先还是使用表输入步骤获得要处理的数据，SQL语句如下：

```
SELECT rental_id
,       customer_id
,       rental_date
,       return_date
FROM   rental
```

然后使用Java脚本步骤的dateDiff函数来计算归还时间和租赁时间之间的星期数。脚本如下:

```
Var age= dateDiff(rental_date,return_date,"w");
```

说明: 和SQL 语句不同, JavaScript是大小写敏感的, 所以写函数时要注意大小写。dateDiff 函数完全不同于datediff 函数。

这里只算出了星期数, 如果星期数大于或等于一周是延期归还, 那么还要增加延期归还的标识字段。这里有几个方法: 使用值映射步骤, 使用公式步骤, 或者再往Java脚本步骤里增加一些代码。如果想尽可能少用Java脚本步骤, 就使用值映射步骤。原因很简单: 如果以后延期归还的条件修改了(如一周内是正常归还, 一周到二周之间是延期归还, 二周以上是超期归还), 使用公式步骤会使逻辑越来越复杂。在这个例子里, 我们使用Java脚本步骤来解决这个问题, 如图7-38所示。

在图7-38里, 只在输出流里增加了status字段, age字段只是用来计算的中间字段, 没有输出。

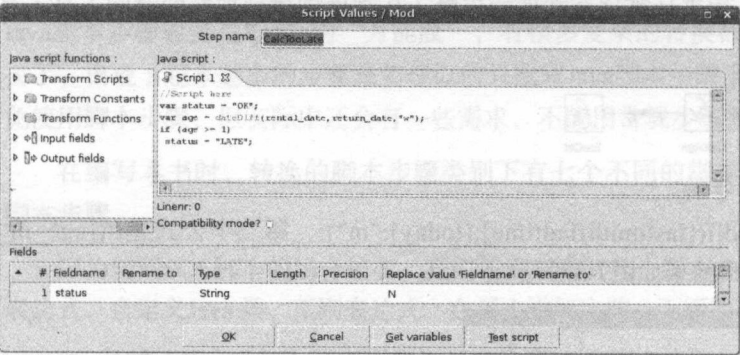


图7-38 计算星期数的Java脚本步骤

7.5.3 用户自定义Java表达式

Java表达式步骤和公式步骤非常类似。唯一不同的是你要使用Java表达式, Pentaho wiki有详细的解释: <http://wiki.pentaho.com/display/EAI/User+Defined+Java+Expression>。

下面是来自Mozilla 公司的一个清洗日志的转换流程, 这个流程演示了如何使用 Java 表达式步骤。因为每个小时要处理几个G的日志文件, 所以要抓住Kettle 里一切能提升性能的机会。图7-39显示了其中的一个清洗流程, 这个流程里使用了三个Java 表达式步骤。

第一个Java表达式步骤使用 url.replaceAll("(^[&])(appID|appVersion)", "\$1&\$2")来替换正则表达式匹配到的字符串。后面两个 “unescape” 表达式使用Java.net.URLDecoder.decode(<urlpart>)方法, 把 Roland,+Jos+%26+Matt这样的字符串解码成 Roland, Jos & Matt。这个转换例子可以从本书网站下载。

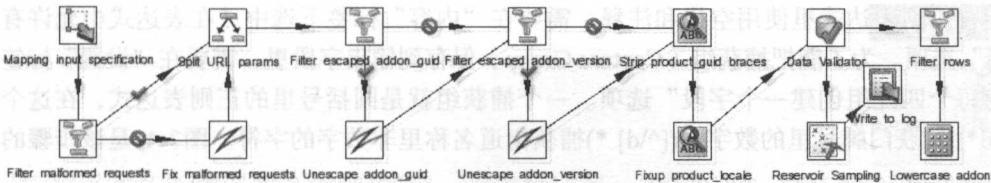


图7-39 在数据清洗里使用 Java 表达式步骤

7.5.4 正则表达式

Kettle 里有很多使用正则表达式的地方，通常是在一些设置通配符的地方，如“文本文件输入”、“获取文件名”或“字符串替换”等步骤。在JavaScript里也可以使用str2RegExp 函数来使用正则表达式。但正则表达式最强大的功能还是要通过正则表达式步骤体现出来。

说明：这个步骤最初的版本只是返回一个Boolean 值来说明正则表达式是否和某个字段匹配。后来，Pentaho 社区的某个成员，又增强了这个功能。可以支持通过捕获组来创建新的输出字段。这个功能和相关的例子都贡献到了Kettle 社区版。这个例子就是/samples/transformations目录下的RegexEval - parse NCSA access log records.ktr文件。Kettle wiki有这个例子的详细解释（<http://wiki.pentaho.com/display/EAI/Regex+Evaluation>）以及正则表达式的教程，帮助你使用正则表达式。

这个例子看上去有点令人眼花缭乱，我们这里看一个简单的例子。首先准备一些数据。我们还使用sakila数据库的地址表，然后使用正则表达式步骤从地址里获取街道名称和门牌号，保存在两个字段里。表输入步骤的SQL语句如下：

```
SELECT address_id
, address
FROM address
```

正则表达式步骤使用地址字段作为“要匹配的字段”，如图7-40所示。

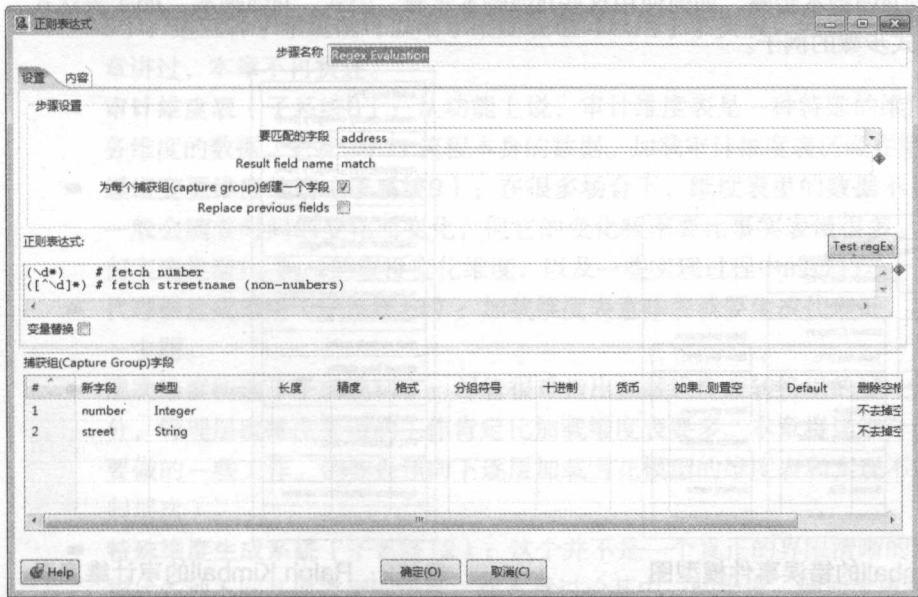


图7-40 正则表达式匹配步骤

为了能在正则表达式里使用空格和注释，需要在“内容”标签下选中“在表达式中允许有空格和注释”选项。为了能把捕获组（Capture Group）保存到输出字段里，需要在“设置”标签下选中“为每个匹配组创建一个字段”选项。一个捕获组就是圆括号里的正则表达式，在这个例子里，(d*) 捕获门牌号里的数字，([^\d] *)捕获街道名称里非数字的字符。图7-41是该步骤的部分结果。

Rows of step: Regex Evaluation (603 rows)

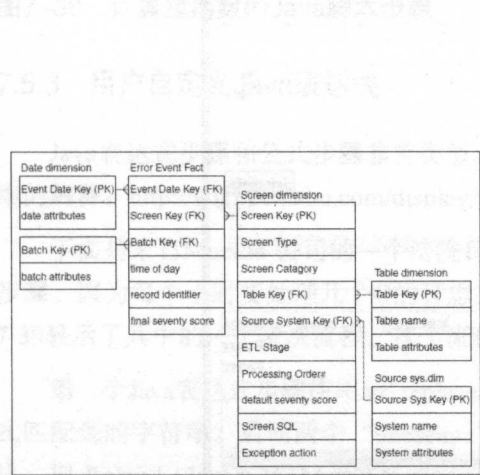
| # | address_id | address | match | number | street |
|----|------------|-----------------------------------|-------|--------|-------------------------------|
| 1 | 1 | 47 MySakila Drive | Y | 47 | MySakila Drive |
| 2 | 2 | 28 MySQL Boulevard | Y | 28 | MySQL Boulevard |
| 3 | 3 | 23 Workhaven Lane | Y | 23 | Workhaven Lane |
| 4 | 4 | 1411 Lilydale Drive | Y | 1411 | Lilydale Drive |
| 5 | 5 | 1913 Hanoi Way | Y | 1913 | Hanoi Way |
| 6 | 6 | 1121 Loja Avenue | Y | 1121 | Loja Avenue |
| 7 | 7 | 692 Joliet Street | Y | 692 | Joliet Street |
| 8 | 8 | 1566 Inegl Manor | Y | 1566 | Inegl Manor |
| 9 | 9 | 53 Idfu Parkway | Y | 53 | Idfu Parkway |
| 10 | 10 | 1795 Santiago de Compostela Way | Y | 1795 | Santiago de Compostela Way |
| 11 | 11 | 900 Santiago de Compostela Redway | Y | 900 | Santiago de Compostela Redway |

图7-41 正则表达式匹配结果

7.6 小结

本章讲述了Kettle里用于数据清洗、校验、更正的几个不同的工具和技术。我们讨论了数据应该遵守的不同类型的规则和约束。在本章里我们学到了：

- 用于数据清洗的不同步骤和这些步骤的用例。
- 如何使用 Kettle 里的字符串匹配算法。
- 用于清洗和更正数据的参照表。
- 如何使用数据校验步骤，校验数据的选项。
- 如何使用Kettle的错误处理机制处理错误。
- 如何使用模糊匹配步骤排除重复数据。
- Kettle里的Java脚本步骤，如何使用这些Java脚本步骤。公式、Java脚本、Java 表达式、正则表达式步骤的例子。



备注1: Ralph Kimball的错误事件模型图
——译者注

| |
|---------------------------------|
| audit key (PK) |
| overall quality category (text) |
| overall quality score (integer) |
| completeness category (text) |
| completeness score (integer) |
| validation category (text) |
| validation score (integer) |
| out-of-bounds category (text) |
| out-of-bounds score (integer) |
| number screens failed |
| max severity score |
| extract time stamp |
| clean time stamp |
| conform time stamp |
| FTL system version |
| allocation version |
| currency conversion version |
| other audit attributes |

备注2: Ralph Kimball的审计维度表
——译者注

第8章 处理维度表

本章将学习如何使用Kettle来管理维度表。尤其要了解Kettle的一些非常有用的功能，使用这些功能可以转换或生成维度表数据，并把这些数据加载到维度表里。在我们介绍详细内容之前，我们先回顾一下第5章讲的34个子系统中提供管理维度表功能的子系统。

- **变更数据捕获（子系统2）**：在做变更数据捕获，把数据加载到非正规化的星型模型中时，有几个需要注意的问题，本章将讨论这一个问题。
- **抽取（子系统3）、数据清洗和质量处理系统（子系统4）和错误事件处理（子系统5）**：这三种子系统既可以用于维度表也可以用于事实表。但这些子系统已经在第6、7章讲过，本章不再赘述。
- **审计维度表（子系统6）**：从功能上说，审计维度表是一种特殊的维度表，它不提供业务维度的数据，它提供ETL流程本身的数据。加载审计维度表已经在第7章讲过。
- **缓慢变更维度处理（子系统9）**：在很多场合下，维度表里的数据不是静态的固定的，一般会随着时间的变化而变化，但它的变化频率要比事实表慢很多。本章描述Kettle如何实现类型1、2、3的缓慢变化维度，以及一些实现过程中的技巧。
- **代理键生成系统（子系统10）**：加载维度表意味着就要生成代理键，本章将深入介绍这一主题。
- **层次维度构建（子系统11）**：尽管很难指出加载维度表转换里的哪一部分是层次管理部分，管理层次维度要做的工作肯定比加载维度表要多。本章将详细介绍处理层次维度需要做的一些工作。例如自顶向下逐层加载雪花模型的维度表和实现不定深度的层次（递归层次）。
- **特殊维度生成系统（子系统12）**：这个并不是一个真正的界限清晰的子系统，一般用来指代各种不同其他类型的维度。在本章讨论其中一些维度，例如生成维度和杂项维度。
- **维度管理系统（子系统17）**：这是本章真正的主题。

8.1 管理各种键

加载维度表的主要部分就是如何管理各种键。主要考虑以下两种键。

- **业务键**：这些键是源系统中的业务主键，用来标识唯一的一个业务实体。
- **维度表代理键**：用来标识维度表里的一行。

说明：有争议的是，还有另外一种键，指向维度表的外键。关于外键可能有些混淆概念，人们经常混淆外键和外键约束。

在本章，外键这个词表示在表里有一列，这一列的值是另一个表的键值，用来把这个表和另一个表关联起来。从这个意义上看，外键只是表里存储了键值的一个字段，键值是另一个表的键。

外键一般是在事实表里，用来指向一个维度表，把事实表和维度表关联起来。事实表在第9章讨论。本章我们讨论雪花模型的时候，可以看到，维度表里也可以有外键，指向其他维度表或桥接表。

在管理数据仓库维度的各类键时，需要做下面一些工作。

- 把源系统中的业务键和维度表里的业务键进行匹配，以判断是否要更新或者插入维度数据。
- 为新的维度行生成代理键。
- 在加载雪花维度表和事实表时，查询代理键。
- 生成日期和时间等维度时，根据属性生成智能键。

对上述每项工作，Kettle都提供了至少一个或者多个转换步骤。在本节，我们先大致介绍一下这些管理键的工作，在本章后面的例子里再详细介绍。

8.1.1 管理业务键

数据仓库里的数据来源于一个或多个业务系统。在源系统和数据仓库系统中，都要确定使用哪些信息可以区分业务实体。如果数据里没有可以用来区分业务实体的信息，你的数据就是没有结构的，混乱而无用的。

源系统中的键

在源系统中，用来区分业务实体的信息称为“键”，对于任意指定的表，键就是一列（或一组列），这一列（或一组列）的数据不能重复。在源数据库里，一般强制要求这个“键”是主键或有唯一约束。这样就避免了源数据库中出现重复数据。

数据仓库里的键

在数据仓库里，业务实体，例如产品、客户等，是以维度表的形式来表现的。这些维度表有它们自己的键，这些键不同于源系统里的键。一般称维度表里的这些键为“代理键”，代理键一般是数据类型为整型的一列，代理键的值和维度表里的属性没有关系，代理键一般是在ETL过程中生成的。

业务键

为了正确加载维度表，需要在维度表里保留源系统中用来区分业务实体的信息，也就是源系统中的键。所以，代理键和源系统中的键都要同时存在于维度表中。一般是把源系统中业务表的主键列保存在维度表中。

在这里，源系统中的键就是指“业务键”，业务键既可以是业务实体自然的“键”，也可以是源系统里本身的代理键。但在数据仓库里，我们把源系统里带来的键，统称为“业务键”。

说明：业务键一般不是维度表的键。数据仓库的一个主要目的就是要维护源系统变化的历史，所以要把相同主键的数据的不同历史版本保留，所以业务主键就会有重复的情况。在后面讨论类型2的缓慢变化维度时，就会看到相关的例子。从性能角度考虑，通常也需要给业务键建索引。

存储业务键

业务键通常直接存储在维度表里。这样ETL过程会比较简单明了。例如，为了检查来源于业务系统的数据应该是更新维度表还是应该插入到维度表，就要通过业务键来检查维度表里是否已经有了相同业务键的数据行。如果有了相同业务键的数据行，通过查询还可以返回该数据行的代理键。

另外，业务键/维度键的映射也可以在维度表之外来维护，在数据缓存（staging area）阶段来维护这个映射关系。在这种情况下，维度表的设计会比较清晰，但是ETL流程会更复杂。

使用Kettle查询键

Kettle提供了几个步骤用来查询数据。这些步骤一般用来查询键（典型的，通过业务键来查询代理键）或者查询属性（一般用来做数据的反规范化）。

除了纯粹的查询步骤，也有几个步骤可以在查询数据的同时，还向数据库中插入和更新数据。在本章和后面的章节中会看到很多这方面的例子。

8.1.2 生成代理键

数据仓库最佳实践表明，原则上，维度表应该使用自动生成的无意义的整型数值作为代理键。但也有一些维度表不必遵循这个原则，在本章后面讨论这方面的例子，但在本节，我们的代理键都遵循这个原则。

Kettle提供了一些功能可以直接在转换里生成代理键，另外也支持通过数据库生成代理键。本节我们学习能生成代理键的Kettle步骤。

“增加序列”步骤

在Spoon里，转换类别下面有“增加序列”步骤，该步骤用来生成自增的整数序列。第4章的“load_dim_date”和“load_dim_time”转换里，我们曾经看到过这个步骤（见图4-4和图4-5）。

这个步骤接收输入流传来的数据，并给输入流里增加了一个类型为整型的字段，然后输出

到输出流。这个字段的字段值可以有以下两个来源。

- 转换自己维护的本地运行时计数器。
- 数据库序列。类似Oracle或PostgreSQL这样的数据库都有序列对象，从统一管理的数据库序列里获取自增整数。

在本节后面我们介绍如何使用“增加序列”步骤生成维度表的代理键。

“增加序列”步骤使用内部的计数器

图8-1显示的是“增加序列”步骤的配置窗口，这个配置使用转换内部的计数器在运行时获取序列值。

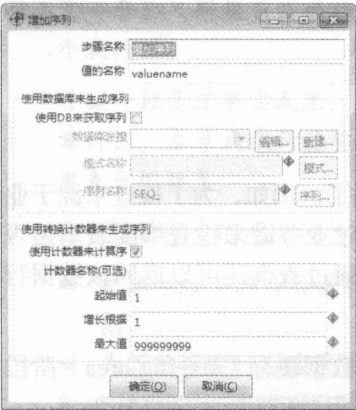


图8-1 配置“增加序列”步骤在转换里生成值

在对话框的上部，在“步骤名称”输入框里设置一个在转换里唯一的名称。在“值的名称”输入框里设置在输出流里序列字段的名称。

在对话框的下部是“使用转换计数器来生成序列”面板，在这里有一个“使用计数器来计算序列”选项，如果选中了这个选项，上面的“使用数据库来生成序列”面板就会不可选，这个选项就指定了序列字段里的值来源于转换运行阶段的计数器。

图8-1的“计数器名称”字段留空。当在一个转换里有多个“增加序列”步骤时，如果在每个“增加序列”步骤里设置相同的“计数器名称”，那么这些步骤使用同一个计数器，这样可以保证每个步骤输出不同的序列值。

“起始值”输入框用来指定序列的偏移。这里可以设置一个整数值，也可以设置一个变量。在图8-1里，这个值设置为1，在后面的一个例子里，我们看如何使用变量来初始化这个偏移。

说明：在这个例子里，如果把一个转换多次运行，每次运行时偏移值还都是一样的。“增加序列”步骤也不会从一个SQL查询里动态地加载偏移值。所以，没有内置的方法可以把偏移值设置为上次转换运行后的序列值。后面讨论的例子将说明如何解决这个问题。

配置对话框里，“增长根据”输入框也要设置为一个整型的数值或变量。用来指定生成的序列的间隔。默认值是1，通常不用修改，只有在一些特定的场合下才可能需要修改。

说明：考虑一个要设置“增长根据”的例子：如果有1到N的多个转换，转换之间都要使用不同的序列值，这时可以从1到N设置每个转换的起始值，而步长统一设置为N。

配置对话框里，在“最大值”输入框里设置序列的最大值。如果计数器超过了这个值，计

数器就会从起始值重新开始。

基于计数器生成代理键

test_sequence1.ktr转换例子使用如图8-1所示的配置的“增加序列”步骤名为test_sequence的表生成一个代理键。每次转换运行的时候，生成100行，每一行都得到一个顺序的代理键。转换如图8-2所示。

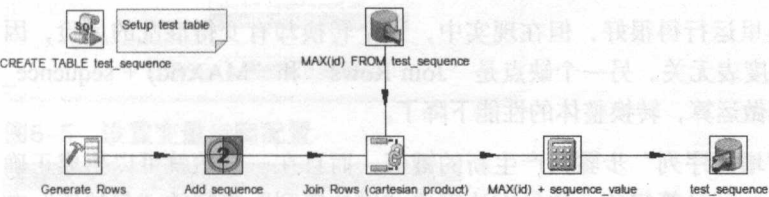


图8-2 使用“增加序列”步骤和内部计数器生成代理键

说明：test_sequence1.ktr转换在本书的例子里。运行这个转换之前检查数据库连接是否匹配你的数据库。转换使用MySQL的test数据库，用户名是test，密码是test。

这个转换运行时，会在转换运行的初始化阶段，先执行“CREATETABLE test_sequence”步骤里的SQL语句。这是一个“Execute SQL script（执行SQL脚本）”步骤，创建了一个表“test_sequence”，用来代表维度表，“执行SQL脚本”里的SQL语句是：

```
CREATE TABLE IF NOT EXISTS test_sequence (  
  id    INTEGER NOT NULL PRIMARY KEY  
);
```

初始化阶段完成后，转换的“Generate rows”步骤生成了100 行空行。这些行代表了要加载到test_sequence表里的数据行。然后是如图8-1所示的配置的“增加序列”步骤，给数据流添加了一个 sequence_value自增序列字段。

sequence_value字段只是一些预备数据，还不能直接作为代理键。每次转换运行时，这些序列的起始值都是1，和数据库里已有数据的代理键重复。

为了解决这个问题，需要把序列值和表里的最大值相加。我们使用一个表输入步骤“MAX(id) FROM test_sequence”来获取 test_sequence表里的最大值。表输入步骤里的SQL 语句如下：

```
SELECT MAX(id)  
FROM test_sequence
```

下一个步骤是“MAX(id) + sequence_value”步骤，这是一个计算器类型的步骤。它把序列和表的最大值ID相加。最终产生一列，可以作为目标表的代理键。计算器步骤如图8-3所示。

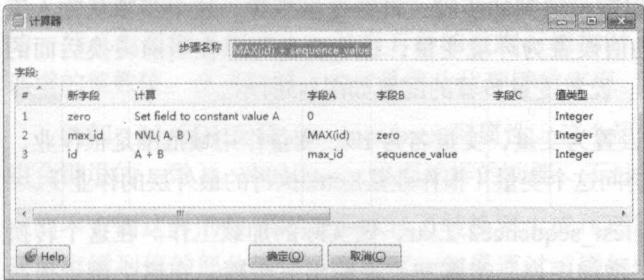


图8-3 使用计算器步骤计算真正的代理键

图8-3的配置也考虑到了目标表是空的情况。此时“MAX(id) FROM test_sequence”步骤会返

回一个NULL值，计算器步骤里的第二个计算公式会把NULL 值替换为 0，以便于后面的计算。

最后是一个“表输出”步骤，用来加载 test_sequence 表，代表了加载维度表。

动态配置序列的偏移

图8-2里的test_sequence转换通过取得数据库里的代理键的最大值，并将最大值和序列相加来解决序列的偏移问题。

尽管这个方法在这个转换里运行得很好，但在现实中，这个转换却有变得混乱的风险，因为有很多步骤和真正的加载维度表无关。另一个缺点是“Join Rows”和“MAX(id) + sequence_value”步骤会给每个数据行都做运算，转换整体的性能下降了。

另外还有一个方法能让“增加序列”步骤只产生新的键值。而且在一开始就可以生成正确的键值而不用一遍一遍地给每一行计算键值。这种方法需要“增加序列”步骤的“起始值”在初始化时是正确的值。

所以要动态配置“起始值”，整个过程不是很直观，但是这种方法使主转换比较清晰，而且会提高一些性能。这个方法就是如图8-4所示的test_sequence2.kjb 转换。

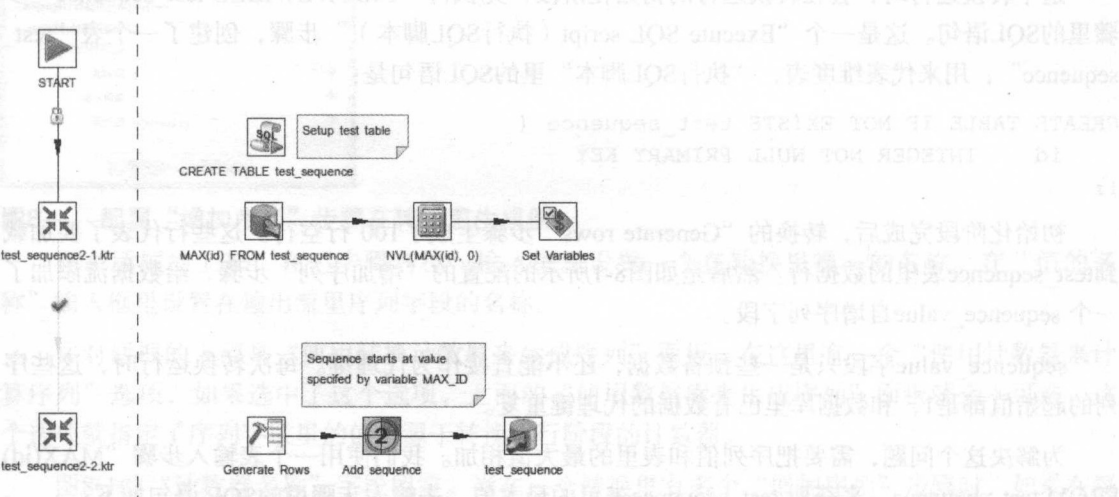


图8-4 在“增加序列”步骤里动态配置偏移

图8-4里的转换，包含两个转换，test_sequence2-1.ktr 和test_sequence2-2.ktr。第一个转换，test_sequence2-1.ktr 创建了test_sequence 表，并从数据库里查到了最大键值，也有计算器步骤最大值是NULL 时把它替换为0。

test_sequence2-1.ktr.转换里还有一个以前没遇到的步骤：设置变量步骤。这个步骤从输入流里接收一行数据，把输入流里某个字段的值设置为环境变量，这个变量可以在当前转换后面的转换中使用。变量作用域的范围可以配置。设置变量步骤的配置如图8-5所示。

在图8-5中，输入流中的id字段的值被设置为变量，变量名为 ID。变量作用域范围是根作业，也就是根作业里的各个作业和转换都可以访问这个变量（根作业是Kettle执行的最外层的作业）。

test_sequence2作业里的第二个转换是test_sequence2-2.ktr，做实际的加载工作。在这个转换里，我们还可以看到“生成行”步骤、“增加序列”步骤和“表输出”步骤。在这个例子里，“增加序列”步骤的配置除了“起始值”不同外，其他的配置和图8-2完全相同。“起始值”设置成了\${ID}，代表对变量的引用，变量是test_sequence2-1.ktr里通过设置变量步骤设置的。这里

的“增加序列”步骤如图8-6所示。

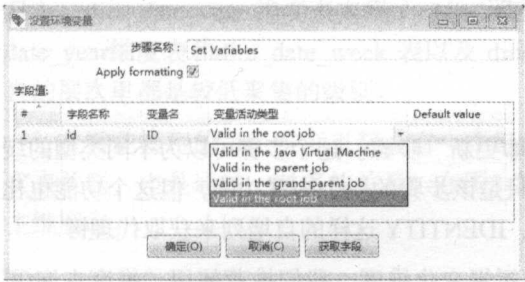


图8-5 设置变量步骤配置



图8-6 在“增加序列”步骤里使用变量

基于数据库序列的代理键

“增加序列”步骤也可以从数据库序列中获取值。在“增加序列”对话框里选中“使用DB获取序列”。

选中一个数据库连接，这个数据库应可以支持序列，然后指定数据库连接本身。如果数据库序列不在数据库的默认模式下，还要指定数据库的模式名。

到这里就设置完了，和设置转换的内部计数器不同，这里不用设置序列的偏移值，如果有请求，数据库就会返回序列的当前值。

处理自增或标识列

一些数据库在列级别上支持序列。在DDL里，使用某些列属性或数据类型，可以创建这样的一列（也可能是多列），每当有新的数据行增加时，数据库会自动为这一列分配一个唯一的递增的整数。这么做的目的就是简化代理键的实现。

例如，MySQL支持auto_increment列属性，这个属性可以用于一个整型的主键列（也可以是组合键里的一列）。SQL Server也有这样的功能，不过是使用IDENTITY数据类型来实现。

一方面这样的数据库功能减轻了转换要生成代理键的负担，另一方面也为转换后面，需要获取自增列值的部分增加了复杂度。如果通过“增加序列”步骤来使用数据库序列，就不会存在这样的问题。在把数据行插入到表之前，就已经获得了数据库的序列值，而且序列值放在转换里，可以便于后面的使用。

可以往维度表里插入数据的Kettle步骤，一般也都有获取自动生成的自增列的值的函数，并将自增列的值加入到输出流中，在本章后面将详细讨论。

缓慢变化维度的键

本章的“缓慢变化维度”详细介绍了“维度查询/更新”步骤。这个步骤可以为不同类型的缓慢变化维度自动生成代理键。尽管自动生成代理键只是该步骤的一个内置功能，但这个功能也比较灵活，它可以根据数据库序列或者auto_increment、IDENTITY 这样的自增列来获取代理键。

如果你要设计一个转换来加载类型1或类型2的缓慢变化维度，我们推荐使用“维度查询/更新”步骤。使用这个步骤就不用再使用“增加序列”步骤来生成代理键。

8.2 加载维度表

在本节，我们看如何使用Kettle把一个或多个源系统或缓存区的数据加载到维度表中。下面看两个典型的场景：

- 加载雪花维度表
- 加载反正规化的星型维度表

8.2.1 雪花维度表

在雪花模型中，一组相关的维度表构成一个单一的维度。这些相关的维度表一般是遵照第三范式设计（3NF）或Boyce-Codd范式（BCNF）设计，也就是“正规化”的结构。图8-7是一个雪花模型的例子，有一个事实表和两个雪花状的维度表：日期维度表和产品维度表。

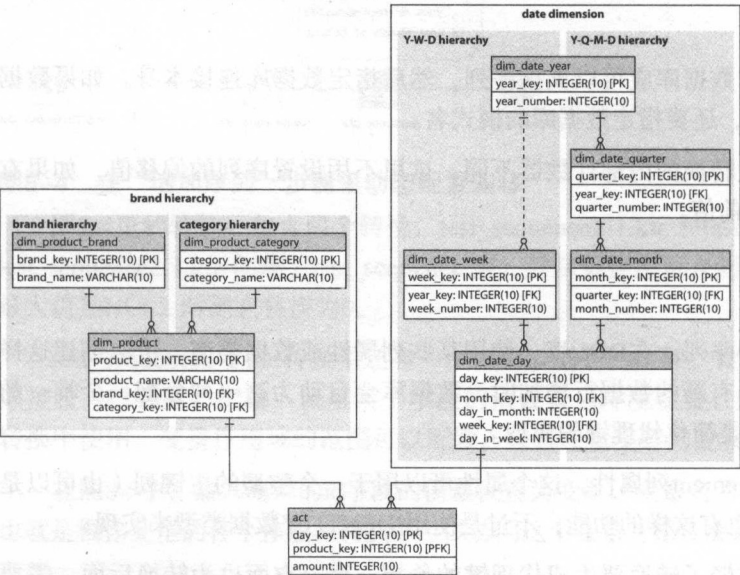


图8-7 雪花维度

一个雪花维度里的每个维度表都是维度层次里的一个级别。维度表之间都是一对多的关系，在一对多关系里，聚集度最高的维度表是一，下面聚集度较低（更细的粒度）的维度表是

多。例如，图8-7中，日期维度有两个层次：Y-W-D（年，周，日）和Y-Q-M-D（年，季，月，日）。dim_date_year 维度表实现了“年”级别，它在两个层次里都是聚集度最高的级别。dim_date_year 维度表和dim_date_week 表以及 dim_date_quarter 表都是一对多的关系。这两个表在各自的层次里都是较低聚集的级别。

在雪花模型的每个维度里都会有一个最低聚集度的维度表：主维度表。主维度表通常和事实表关联，也是通过一对多的关系。在图8-7里，dim_product和dim_date_day分别是各自维度的主维度表。

自顶向下的逐级加载

在加载雪花模型的维度时，除了顶级的维度表（聚集度最高的维度表）外，加载其他维度表时，都要查询它上一级的维度表。查询是为了获取上级维度表的代理键，把上一级维度表的键保存到当前维度表里，以保持两个维度表之间的关系。

因为低层维度表在加载时要依赖与之相临的高层维度表，所以通常使用自顶向下的加载层次，先加载最高层次的维度表，然后加载最高层下面一层的维度表，然后再下一层，依此类推，最后加载主维度表。这就是我们所说的自顶向下的逐级加载。

说明：我们前面遇到过另一个例子，它的加载顺序也依赖于查询顺序：第4章的load_rentals 作业，这个作业在加载事实表之前，使用了一系列的转换作业项来加载维度表。

在处理这种自顶向下逐级加载的雪花模型时，一种较好的方法是把每个单独的维度表放在一个转换里。每个转换都抽取该维度表所代表层次的增量数据。如果要加载的维度表不是维度的顶层，还需要使用一个或多个查询步骤从上一层级中获取代理键。

sakila雪花模型例子

为了演示整个过程，我们创建了sakila_snowflake库。它是第4章中介绍的sakila 样例数据库的另一种维度模型。除了customer和store维度，sakila_snowflake 库和前面介绍的租赁星型模型完全一样。customer和store这两个维度的维度表dim_customer和dim_store 里所有和地址相关的列都变成一个外键，指向雪花状的地址维度的主维度表。sakila_snowflake 库（部分）如图8-8所示。

雪花状的地址维度包括三个表：dim_location_address、dim_location_city和dim_location_country。dim_location_address表是主维度表。代表最低层的地址维度。

说明：sakila_snowflake例子的安装过程与租赁星型模型的安装过程相同。从本书网站下载sakila_snowflake_schema.sql和create_sakila_snowflake_accounts.sql 文件。

要了解加载过程，还要下载 load_dim_location_%作业和转换文件。

在sakila_snowflake模型里，地址维度的主维度表并没有和事实表关联，而是和其他维度表关联。因为地址数据是从已有的store和customer 维度表中抽取出来的，这些地址信息被组织成雪花状。在Ralph Kimball和Margy Ross的*Data Warehouse Toolkit, Second Edition*（Wiley 2002）一书中，Kimball 也认为这是一种有效的地址维度正规化的方式，他把这种结构称为“地址支架”（尽管他不推荐把地址维度本身再组织成雪花状）。因为这个模型的大部分还是星型模型，所以我们也把这种模型称为“星型雪花”模型。

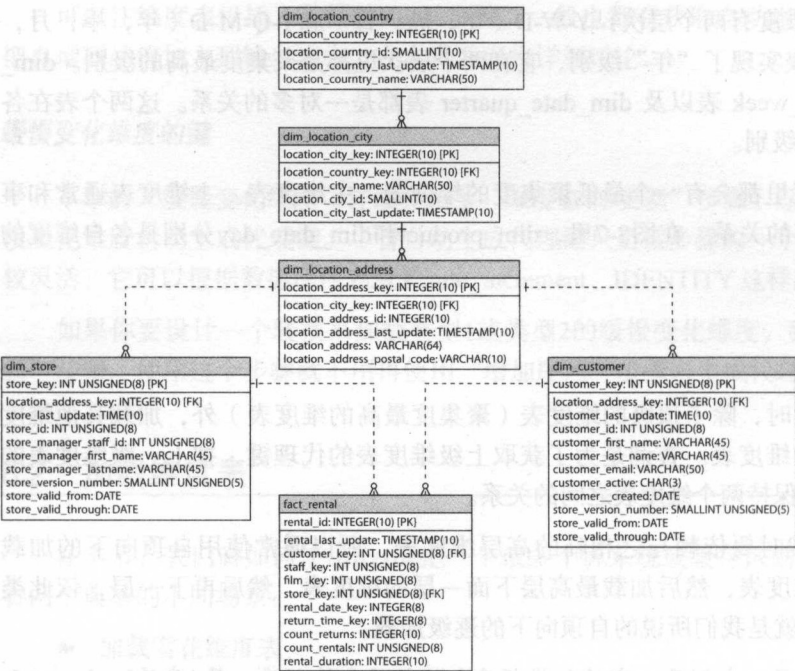


图8-8 sakila_snowflake 模型（部分）

转换样例

load_dim_location_address.ktr转换如图8-9所示。



图8-9 加载雪花模型中的某个单独维度表

这个转换用来加载图8-8所示地址维度里的dim_location_address 表。前两个表输入步骤用来抽取增量数据，这种方法在第4章介绍过：先在维度表里查询last_update列里最大的时间，然后使用这个时间从源系统中抽取增量数据。

抽取完增量数据后，把数据传到“Lookupdim_location_city”步骤，再从dim_location_city表中查找对应的代理键。我们后面再讨论这个步骤的配置。

最后，使用“插入/更新”步骤把地址数据以及指向dim_location_city表的外键都加载到dim_location_address 表中。在后面的“类型I缓慢变化维度”一节中再详细介绍这个步骤。

数据库查询配置

在图8-9的load_dim_location_address.ktr转换里，使用了一个“数据库查询”步骤来找到和源

系统业务键（city_id）相对应的数据仓库里的代理键（location_city_key）。

在本章的后面可以看到，“数据库查询步骤”也可以用来抽取源系统里的属性值，这些属性值都用于数据的反规范化，并加载到维度表中。在第9章，当加载事实表时，数据库查询步骤还用于查找维度表的代理键。

图8-9中load_dim_location_address转换里的Lookup dim_location_city步骤的配置如图8-10所示。

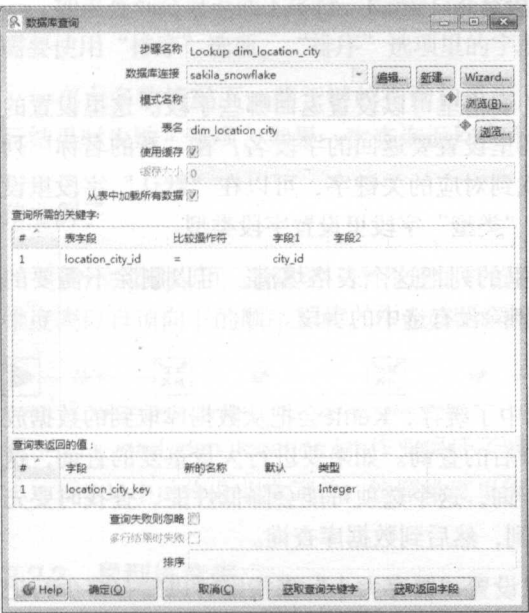


图8-10 “数据库查询”步骤的配置

连接、查询模式和查询表

图8-10里的配置非常简单：因为要查询dim_location_city 表里的代理键，所以这里填入相应的连接名称和查询表名称。

对于“数据库连接”输入项，可以从下拉列表中选择一个已有的连接。也可以使用“编辑”按钮预览或修改已经定义好的连接。

在本例里，“模式名称”输入项可以留空，sakila_snowflake是连接的默认模式，但如果需要也可以指定其他模式，可以直接敲模式的名字，或者可以通过“浏览”功能选择模式。

要设置“表名”输入项，来指定从哪个表中进行查询。同样可以直接输入，或者通过“浏览”功能选择表。

查询所需的關鍵字

对话框的下部是两个表格。第一个表格的标题是“查询所需的關鍵字”，用来指定输入数据流和查询表里的哪几列是查询的关键字。另外表格里的每一行还定义了比较条件（有的比较条件需要输入流里的两个字段）。

表格里的第一列是“表字段”。在这里，可以指定查询表里的字段名，一般来说，这里设置的字段名，是查询表的键。数据流里的每一行都可以匹配到查询表里的至少一行。例如，图8-10里指定的location_city_id字段是dim_location_city_table 表的业务主键。

表格里的“字段1”，是指定使用数据流里的哪一列来和“表字段”比较。“比较操作符”字段指定表字段如何和数据流里的字段进行比较，大多数情况都使用“=”比较符。

有时候，要查询一个取值范，就要使用“BETWEEN”操作符。此时要指定流里的两个字段，“字段1”和“字段2”。

如果想快速填写表格里的内容，可以单击对话框下方的“获取查询关键字”按钮，把输入流里的每个字段名自动填充到表格里，操作符自动设置为“=”。此时如果表里的字段名和流里的字段名不同，要手工修改表格里自动填充的表字段名。

查询表返回的值

在对话框下部的标题为“查询表返回的值”的表格里可以设置返回哪些字段。这里设置的每个字段都加入到输出流里。在表格的“字段”列里设置要返回的字段名，在“新的名称”列里可以把要返回的字段改名，如果在表里没有查找到对应的关键字，可以在“默认”字段里设置返回字段的值。当指定了“默认”字段，还要在“类型”字段里设置字段类型。

通过“获取返回字段”按钮可以使用查询表里的列把这个表格填满。可以删除不需要的列，或者选中需要的列，然后按Ctrl+K组合键，来删除没有选中的字段。

缓存配置

“数据库查询”步骤里可以配置缓存。如果选中了缓存，Kettle会把从数据库取到的数据放到内存的缓存里，然后使用内存里的数据去处理以后的查询。如果要进行大量重复的查询，这种方法会极大地提高性能。如果只有少量的重复查询，这个选项可能会降低性能，查找时先要在内存中查找一遍，而大部分数据在内存中查找不到，然后到数据库查询。

在图8-10里，选中“使用缓存”功能后，可以设置“缓存大小”来指定可以缓存多少行记录。如果缓存里的数据行数达到了最大值，当有新的数据行要写入到缓存时，最早的数据行会被删除。如果把“缓存大小”设置为0，所有的查询结果都会缓存。

图8-10里的“缓存大小”被禁用了，说明没有使用这个设置。这是因为我们选中了“从表中加载所有数据”选项。这个选项强制从dim_location_city表中加载所有的数据行作为缓存。如果没有选中这个选项，缓存的处理机制是：每当有一个数据查询的请求，先从缓存里查找数据行，如果缓存里没有，再从数据库表里找，如果找到了，把找到的记录再放到缓存里，以后如果有相同键值的查询，就会提高查询速度。

当选中了“从表中加载所有数据”选项，在转换的初始运行阶段，查询表只会被扫描一次，表中所有的记录都被保存到缓存里。在运行时，所有的查询都在缓存里完成，不用再去访问数据库表。这样就不用再花时间去维护缓存和执行数据库查询，所以这个选项的性能会比较高。但如果查询表太大了，会消耗非常多的内存，如果没有足够的内存就不能选中该选项。

查询失败

我们前面讲“数据库查询”步骤，都假设查询会成功，也就是在表中有查询对应的记录，而且只有一条记录。但很多场合下并非如此，查询表里可能没有要查询的记录，或查询表里有多行要查询的记录。

通常，如果没有在查询表里找到要查询的记录，所有的字段就使用在“查询表返回的值”里设置的字段默认值，如果没有设置默认值，就返回NULL值。但如果想丢弃那些在查询表中没有查询到的记录，可以选中“查询失败则忽略”选项。

说明：在大多数情况下，由于查询失败导致的NULL值，迟早都会引起问题，可能会使整个转换处理失败。例如，图8-9的load_dim_location_address转换，因为dim_location_address表要求location_city_key列不能为NULL，所以如果没有查询到，加载表的时候转换就会失败。

转换失败也是一件好事：我们前面解释过，自顶向下，逐级加载的转换要求在加载dim_location_address表之前dim_location_city表已经被加载完成。所以查询失败可能意味着应用设计或源系统或目标维度表有逻辑缺陷。

如果查询结果返回了多行，Kettle只获取返回结果的第一行。如果想让Kettle获取指定行记录，需要使用“排序”选项。“排序”选项里的字段将被放入到SQL的“ORDER BY”子句后面。

在大多数情况下，基于键的查询如果返回多个结果说明有逻辑问题。此时，可以选中“多行结果时失败”选项，如果一次查询返回多个查询结果，步骤以及整个转换都会执行失败。

作业例子

设计完转换后，还需要一个作业把转换顺序组织起来，来保证雪花模型一个维度里的各个维度表以自顶向下的顺序加载。作业如图8-11所示。



图8-11 load_dim_location.kjb作业保证了雪花模型维度表的自顶向下逐层加载

图8-11中的作业非常简单。按照维度里级别的顺序加载地址维度里的各个维度表。

8.2.2 星型维度表

在星型模型里，使用单一的一个维度表来表示一个维度（如图4-2所示的租赁星型模型）。维度里的层次由表里的各个列组成，每个列就是层次里的一个级别。

反正规化

通常，星型模型的维度表都非常宽，有很多列，正如Kimball所说，这些列一般都是“高度相关”的。也就是说在功能上这些列里的值是互相依赖的。因为列之间互相依赖（不仅仅依赖于维度表的键），所以这些维度表并不遵循第三范式，甚至也不遵循第二范式（3NF和2NF）。星型维度模型表里还会有多值列，多值列是值的一个列表，代表源系统中的多个实体，因此还违背了第一范式。因此，无论怎么分类，星型模型的维度表都是“非正规化的”。

在很多情况下，维度表里的数据来源于OLTP源系统，OLTP源系统底层的数据库系统一般遵循第三范式或Boyce-Codd范式（BCNF）。所以，为了加载星型模型表，就需要从多个相关的源系统表里抽取数据，把记录连接起来，也就是把数据进行“反正规化”处理，然后就可以把数据加载到目标维度表里。

使用“数据库查询”步骤反正规化到第一范式

在前面讨论的加载雪花模型的例子里，我们详细描述了“数据库查询”步骤，它可以使用源系统的业务键来查询维度表的代理键。但是“数据库查询”步骤也可以通过连接源系统中的

数据,来完成数据的反正规化的工作。“数据库查询”步骤只是简单地使用数据链中的键值来查询,它不关心返回的只是一个代理键还是整个一条记录。

在第4章中,我们看到了几个使用“数据库查询”加载星型模型维度表的例子。第一个例子是fetch_address子转换(图4-14),它通过级联查询从sakila库的地址、city和country表中抽取地址数据,形成了dim_customer和dim_store维度表里的地址列。

我们这里不再重复介绍“数据库查询”步骤的功能,但为了完整性和比较的目的,建议读者还是要花一些时间比较“数据库查询”步骤在第4章里的使用方法和在本章前面雪花模型里的使用方法。

增量数据捕获

在第4章讨论的星型模型里,见图4-2,每个维度表都是基于sakila 库里的一个表。例如, dim_store 维度表基于store 表, dim_customer维度表基于 customer 表,等等。为了加载维度表,需要从对应的表里抽取增量数据,然后使用“数据库查询”步骤反正规化,直到能加载到维度表里。

如果把第4章的增量数据捕获的过程和本章雪花模型的逐层加载的过程相比,会发现一个明显的不同。第4章的转换只能检测到维度表底层的那些数据的变化,然后再用底层的数据去找上一级的数据,但是如果上一级的数据发生了变化了怎么办?那些上一级变更的数据不会被捕获到,源系统和目标系统就会不一致。不过有几个方法可以确保维度里各级的数据都可以被捕获到。

如果维度表不太大(维度表通常都不大),不用考虑复杂的方法去做增量数据抽取。直接遍历底层维度表所对应的源系统中表的所有数据行,级联查询上一级的数据,上一级的变化数据就会自动包含在最终的反正规化的结果集里。

如果必须要从源系统中抽取变化的数据,加载星型模型是一件比较棘手的事情。如果这么做的目的只是为了减少源系统的负载,那么还有一种相对简单的方法:利用缓冲区。只从源系统中抽取每个表的增量数据,不用做任何变化,把这些增量数据加载到和源系统结构一样(正规化结构)的缓冲区中,缓冲区也是ETL系统的一部分。在缓冲区里我们再使用上面全表遍历的方法,对源系统没有任何影响。

说明:也可以不使用缓冲区来加载变化的数据。但是转换就会变得比较复杂。把源系统的增量数据直接加载到反正规化的星型维度表里是一个高级主题,超出了本章范围。在这里只简单介绍Kettle如何能满足这样的需求。

例如,要加载一个包含了街道、城市、国家等级别的维度,假设捕获到的增量数据也有这些级别,类似我们加载雪花模型里的地址维度。但对于反正规化的维度来说,必须要自顶向下,先找到最高级别的变化数据,然后再找到下面级别与变化数据关联的数据。通过级联查询最终得到一个反正规化的结果集。

在Kettle里可以使用“数据库连接”步骤,从高级别变化的数据“向下找”相应的数据,如第4章的load_dim_film转换(见图4-16为dim_film_actor_bridge表创建标记部分)。但是这种方法非常笨重,因为当捕获到较高级别表的变更数据后,要向较低级别的表发出大量的查询。有的时候直接在表输入步骤里写JOIN类型的SQL查询更有效。

我们前面还看到了一种方法来解决这种复杂的加载问题:使用正规化的或雪花维度,这样

问题就基本不存在了。另外一种方法就是使用第19章的建模方法，Data Vault 架构，我们后面再讨论。

8.3 缓慢变更维度

本节我们详细讨论如何使用Kettle实现不同的缓慢变更维度（SCDs，也简称为缓慢变更维）。我们简要描述每一个缓慢变更维度的特点，然后看使用Kettle的哪个步骤可以加载缓慢变更维度。

说明：本书不提供数据仓库理论的基本知识，如果想知道缓慢变化维度的目的或者你不了解在数据库层次上实现缓慢变化维度的通用技术，要先阅读一些数据仓库方面的图书，例如Ralph Kimball和Margy Ross编写的*The Data Warehouse Toolkit, Second Edition*，Roland Bouman 和Jos van Dongen编写的*Pentaho Solutions*一书的第7章里也有不同类型的缓慢变更维度的目的和特征。

8.3.1 缓慢变更维类型

根据Kimball的理论，有三种类型的缓慢变更维：类型1、类型2、类型3。

- 类型1：对源系统的更新，也会直接更新目标的维度表。
- 类型2：对源系统的更新，会往目标维度表里插入一行数据，通过不同的时间戳来维护同一条维度数据的多个版本。在任何一个给定的时间点，都可以找到一行对应的维度数据。
- 类型3：对源系统的更新，会在目标维度里增加列，在目标维度表同一行新增的列里保存新的数据。

8.3.2 类型1的缓慢变更维

在类型1的缓慢变更维里，维度表总是保存当前的状态，如果发生变化就直接覆盖。有几个Kettle步骤可以直接用于这种类型的维度。

在后面的“类型2的缓慢变更维”里，我们讨论“维度查询/更新”步骤，它可以用来加载不同的缓慢变更维，也包括类型1。在本章的后面的“杂项维度”里，将讨论“联合查询/更新”步骤。本节剩下的部分讨论“插入/更新”步骤，该步骤用于类型1的缓慢变更维。

插入/更新步骤

插入/更新步骤在Spoon界面左侧的步骤类型树的“输出”类别里。正如这个步骤的名字，该步骤可用于插入，也可用于更新数据，我们也称其为插入更新（upsert）。在第4章的load_dim_actor 转换里遇到过这个步骤，让我们再仔细看一下load_dim_actor 转换里的配置（如图8-12所示）。

如果把图8-12和图8-10 的数据库查询步骤比较一下，可以看到很多相似的地方。和“数据库查询”步骤一样，“插入/更新”步骤也使用数据流里的某些字段的值和数据库里的某些字段的值做比较。这些字段一般都是查询键字段。如果数据库表里有匹配到的记录，则使用数据流里的数据去更新表里的数据，如果没有匹配到的记录，则把流里的数据插入到表里，作为一条新的记录。

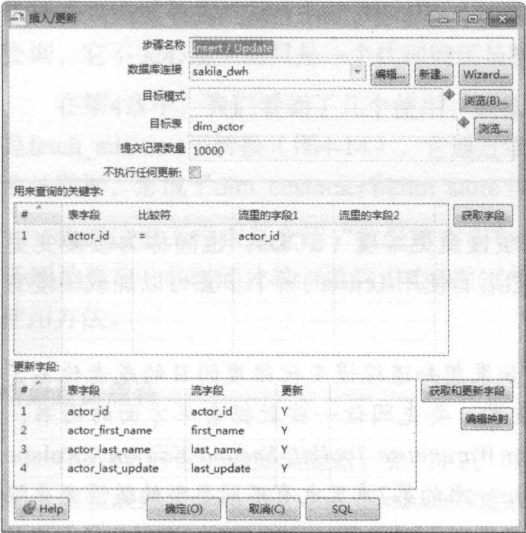


图8-12 插入/更新步骤的配置

说明：如果你只是想插入数据，而且并不担心主键或唯一约束的冲突，就不要使用插入/更新步骤。应该直接使用“表输出”步骤。可以给“表输出”步骤定义错误处理，来捕获插入过程中的错误。这个方法不必花费时间去查询，所以要比“插入/更新”步骤快。

下面讨论图8-12中的一些配置项。

连接，目标模式和表，提交记录数量

在图8-12对话框的上半部分，可以看见有几个配置数据库信息的输入项，包括数据库连接、表模式和表名，这些和“数据库查询”步骤相同。唯一不同的是输入项的名称，这个步骤的名称是“目标模式”和“目标表”，“数据库查询”步骤是“查询模式”和“查询表”。

“提交记录数量”用来指定一次批量提交的记录数。就是在执行COMMIT 命令之前，向服务器发送的记录行数（INSERT和UPDATE 语句的数量）。这个值越大，COMMIT 命令的个数就越少（每次提交的记录就越多）。通常这个选项都是因为性能问题才设置的。但这个值不能太大，一个事务里未提交的数据越多，消耗的数据库资源就越多，反而会影响数据库的性能。

我们不能给出“提交记录数量”参数的建议值。因为这个参数值依赖于很多因素，例如数据库类型、可用内存数、在加载维度时数据库的活动状态。但是默认的数值100 显得比较保守，可能还会影响性能。在很多情况下，可以使用1000甚至10000。关于调整这些配置选项的更多信息可参考第15章。

对话框里还有一个选项“不执行任何更新”。默认情况下，这个选项不选中，这样这个步骤可以做更新和插入。如果选中了这个选项，不会做更新操作，只会做插入操作。

关键字

和“数据库查询”步骤一样，“插入/更新”步骤也要指定查询的关键字。在关键字里设置的字段将会用在UPDATE 语句的WHERE 子句里。

在“数据库查询”和“插入/更新”步骤里，关键字字段都定义在“用来查询的关键字”表格里。除了表格里的设置数据流的列标题不太一样之外（“插入/更新”步骤里是“流里的字段1”，“数据库查询”步骤里是“字段1”），这两个步骤里关键字段的设置基本一样。我们也

可以使用“获取字段”按钮自动填充表格。

在前面图8-10的“数据库查询”步骤里，步骤使用了业务主键来匹配源系统里的数据行。同样，在这个例子里使用dim_actor 维度表里的actor_id列去匹配源系统的actor_id字段。

更新字段

图8-12对话框的下部也是一个表格，表格标题是“更新字段”。在这里设置用数据流里的哪些字段去更新表里的字段或把这些字段插入到表里。

在表格的右侧，有两个按钮，“获取和更新字段”按钮用来把数据流里的字段填充到这个表格里，“编辑映射”按钮可以打开一个向导来简化流字段和表字段的映射工作。映射对话框如图8-13所示。

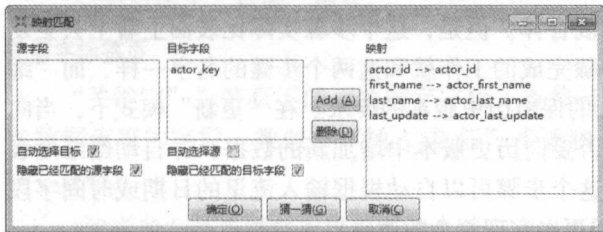


图8-13 字段映射向导可以简化流字段和数据库字段的映射工作

对话框有3个列表：左侧的一个是“源字段”列表，其中是数据流里的字段；中间的是目标表里的“目标字段”；右侧的是“映射”列表，是一组映射，表示流里的哪个字段对应到目标表里的哪个字段。选中“自动选择目标”复选框后，如果选中了数据流里的字段，向导会自动选择一个表里的字段。类似的，如果选中了“自动选择源”复选框，那么若选中表里的字段，向导会自动选择一个数据流里的字段。

同时选中流里的和表里的一对字段名后，通过“Add”按钮，可以把选中的这对字段名映射添加到右侧的“映射”列表里。单击“猜一猜”按钮，系统可以快速自动匹配，并把结果填充到右侧的“映射”列表里。如果选中了“隐藏已经匹配的源字段”或“隐藏已经匹配的目标字段”复选框，那些已经匹配好的字段会从对应的列表中移除。单击“删除”按钮，会删除右侧“映射”列表里的某个映射。列名会重新出现在对应的列表里。

8.3.3 类型2的缓慢变更维

类型2缓慢变更维的特点是它可以按照时间跟踪到维度的变化。类型1的缓慢变更维不会保留历史数据，新的数据会把旧的数据覆盖，而类型2缓慢变更维每当数据更新，会在维度表里增加一行。通过这种方式，类型2缓慢变更维保存了维度的一组不同版本，这些同一个维度的不同版本有相同的业务键。

在第4章中，我们看到了几个类型2的缓慢变更维度：dim_customer、dim_staff、dim_store。在第4章介绍的几个转换里，我们使用了专门为类型2的缓慢变更维度而设计的步骤：“维度查询/更新”步骤。在转换load_dim_customer（图4-11）和转换load_dim_diff（图4-8）中使用了这个步骤，用来加载维度表。我们也在load_fact_rental转换中使用了这个步骤（图4-18），用来查询这些维度表的键值。在本节，我们详细描述如何使用这个步骤加载维度表。在第9章，我们再描述如何在加载事实表时，使用这个步骤去查询维度的键值。

“维度查询/更新” 步骤

在Spoon里，“维度查询/更新”步骤位于“数据仓库”类别下。这个步骤的配置对话框如图8-14所示。在这个例子里，我们使用load_dim_customer 转换里的配置（图4-11）。

“维度查询/更新”步骤可以用于两种不同的模式：

- 可以用于增加或更新维度表里的数据。这个功能用于类型1和类型2缓慢变更维。这个模式也被称为“更新”模式。
- 可以用于查询步骤，用来抽取类型2缓慢变更维的代理键。这个功能在加载事实表时非常有用，这个模式也被称为“查询”模式。

因为“维度查询/更新”步骤把这些不同的功能合成到了一起，所以从表面上看它就像是“插入/更新”步骤和“数据库查询”步骤的混合体。但是，这个步骤实际比表面上看上去更复杂。“插入/更新”步骤和“数据库查询”步骤完成的工作就和这两个步骤的名字一样，而“维度查询/更新”步骤可以满足类型2缓慢变更维的保留历史版本的要求。在“更新”模式下，当向维度表里加载数据时，这个步骤可以判断是否要向历史版本中增加新的数据，并自动给维度表里的版本控制列赋值。在“查询”模式下，这个步骤可以自动根据输入流里的日期或时间字段从维度表里获取适当版本的维度记录，而不用再去实现复杂的逻辑。

我们也可以使用Kettle的一组其他步骤来实现与“维度查询/更新”步骤同样的功能。但是，这样的转换非常复杂，需要为每一个类型2的缓慢变更维都使用这组步骤，这样转换将变得难以维护。

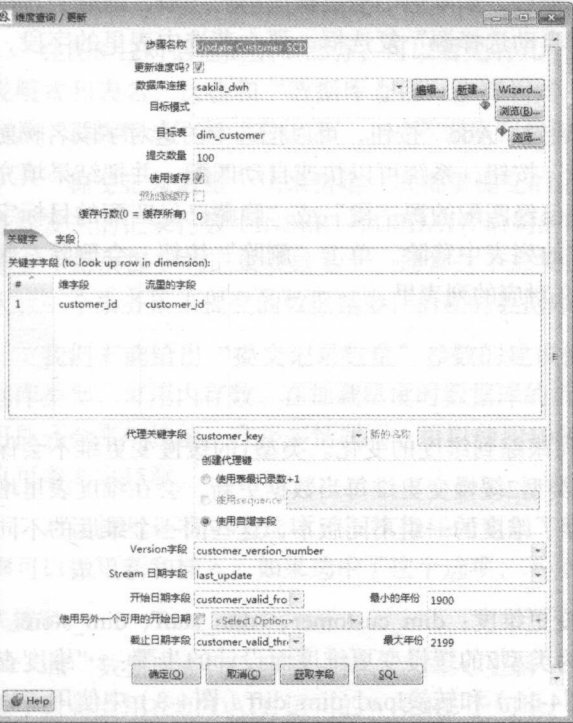


图8-14 load_dim_customer 转换里的“Load dim_customer SCD”步骤的配置

指定操作模式

“更新维度吗”复选框用来指定操作模式。如果选中，就是加载维度表的“更新”模式，

如果没有选中，就是“查询”模式。在后面，我们主要描述“更新”模式。在第9章加载事实表时再描述如何使用“查询”模式。

通用配置项

“维度查询/更新”步骤的通用配置项和“数据库查询”步骤（图8-10）以及“插入/更新”步骤类似（图8-12）。

- “数据库连接”、“目标模式”、“目标表”、“提交数量”和“插入/更新”步骤里的设置类似。注意“提交数量”选项只适用于“更新”模式。
- “使用缓存”、“预加载缓存”、“缓存行数”属性和“数据库查询”步骤里的“使用缓存”、“从表中加载所有数据”、“缓存大小”属性含义相同。“预加载缓存”选项只适用于“查询”模式。

关键字标签页

“关键字”标签页下的“关键字字段（查找行维度）”表格用来设置维度表里的业务主键和数据流里的字段。类似于“插入/更新”步骤和“数据库查询”步骤里的“用来查询的关键字”表格。

“维字段”是指维度表里的业务主键。“流里的字段”是指数据流里的字段，用来匹配维度表里的业务主键字段。只能使用“等于”来匹配，所以在表格里就没有设置比较类型的选项。

在进行关键字匹配时，“维度查询/更新”步骤与“插入/更新”、“数据库查询”步骤还有一个重要的区别。在类型2的缓慢变更维里一个业务主键会有多条记录，保存不同时间的历史版本。因此，需要一定的策略从这多个版本里挑出一个正确的版本，在后面的“历史维护”部分再详细介绍。

代理键

“维度查询/更新”步骤里有几个与代理键相关的设置项，都位于图8-14对话框中的下半部分。

“代理关键字段”用来指定维度表主键的字段名，例如图8-14中，设置为customer_key的字段。“创建代理键”栏中的一组字段用来设置如何生成代理键的值。这些仅在插入一行数据时才有用，所以仅用于“更新”模式。可以选择下面三种方式之一来生成代理键。

- 使用表最记录数+1（最大值+1）：正如这个名字，查找出代理键字段的最大值，然后加1，作为新数据行的代理键。
- 使用sequence：指定一个数据库序列的名称。该选项需要数据库支持序列。（见本章前面的“基于数据库序列的代理键”），这里的序列应该位于数据库连接的默认模式下，如果不在默认模式下，需要上面指定“目标模式”。
- 使用自增字段：如果维度表的代理键使用的是自增列，如auto_increment 或 IDENTITY 列，可以选择这个选项（见本章前面的“基于数据库序列的代理键”）。

历史维护

“维度查询/更新”步骤里还有一组配置项用来配置如何处理类型2缓慢变更维的历史版本数据。这些配置项都在对话框的下部。

“Version字段”用来指定维度表里版本字段的名称，这个字段存储相同业务键的每行记录的版本号。业务主键和版本号字段可以唯一标识出维度表里的一行。在“更新”模式下，如果插入一行新的数据，“维度查询/更新”步骤会自动生成一个版本号。

“Stream日期字段”用来指定流里的日期字段, 这个字段提供了维度变化的时间先后顺序。例如, 在图8-14里, 这个属性可设置为last_update字段。你可能还记得在load_dim_customer转换里, 这个字段来源于customer表的last_update字段。这个字段是一个时间戳, 用来说明在源系统中记录的最后变化时间。

说明: 尽管customer表的last_update字段看上去非常适合于“Stream日期字段”设置项, 但实际上这个字段并不完全合适。因为客户维度表是一个反正规化的维度表, 因此包含了源系统中的多个表。这个内容在我们前面的星型模型的增量数据捕获部分曾经提到过。

尽管sakila数据库的customer表是客户维度层次里底层的级别。但它并不是构成反正规化的dim_customer维度表的唯一的源系统中的表。如果客户维度里更高级别的数据发生了变化, 如客户的地址, 也应该在维度表里生成一条新的记录。此时应该从address表的last_update字段里获取变更时间。

所以为了保证反正规化后的数据行的完整变更顺序, 我们需要把所有构成维度表的业务表里的last_update字段值取出来, 然后从中选择一个。应该从这些last_update字段值里选择时间最近的值, 因为可以确定, 在那个时间点, 反正规化后的维度表记录是存在的。

在很多现实场景中, 源系统中并没有last_update字段。此时, 必须规定一个时间顺序。最常用的方法就是使用当前时间戳。因为实际变化时间是在发现变化并加载到数据仓库之前, 所以可以使用当前时间戳。在这种情况下, 就不用再设置“Stream日期字段”输入项: 步骤会自动使用系统时间作为变化时间。

类型2的缓慢变更维的维度表里还有一对时间字段, 用来指定维度的有效期。在对话框里的“开始日期字段”和“截止日期字段”输入框里可以设置这对时间字段。在图8-14中, 这对字段分别设置为customer_valid_from和customer_valid_through。使用这对字段, 可以有相同业务主键的维度记录里找到合适的一个。这就是“维度查询/更新”步骤的匹配过程和“数据库查询”、“插入/更新”步骤的匹配过程的区别。“维度查询/更新”步骤除了使用业务主键外, 还使用“Stream日期字段”来做查询, 查找业务主键相等而且“Stream日期字段”里的日期介于维度表里开始日期和截止日期之间的维度记录。

对于某个业务主键里写入的第一条记录, 需要指定一个初始日期范围, 这个日期范围可以在“最小的年份”和“最大年份”输入框里设置。默认情况下, 这两个输入框里的值分别是1900和2199, 所以当一个新业务主键的记录被写入到数据库时, 它的“开始日期字段”的值是1900-01-01, “截止日期字段”的值是2199-12-31。默认值是一个很大的日期范围, 一般维度表的历史数据不会超出这个范围。

如果维度表里已经有了相同业务主键的记录, 数据流里的记录会和维度表里相同业务主键的记录进行比较。如果检测到发生了变化, 已有记录的“截止日期字段”的值会变更为数据流里“Stream日期字段”的值。另外, 数据流里的这条记录会插入到维度表里, 作为一个维度版本。对于新插入的这条记录, 它的“开始日期字段”的值是“Stream日期字段”的值, “截止日期字段”的值是已有的上一条记录的“截止日期字段”的值。最后, 新插入的维度版本和维度表中已有的维度版本有连续的日期范围, 这样可以追踪同一个业务主键的不同变更历史。

对于相同业务主键的第一条记录, 除了设置固定的一个日期外, 也可以使用其他的更有实际意义的开始日期。选中“使用另外一个可用的开始日期”复选框, 然后从下拉列表里选择日期。这个下拉列表提供了下面几种日期: “转换开始日期”、“系统日期”、“NULL”或“一

个列值”。例如你想把第一条记录的开始日期设置为“Stream日期字段”的值，可以在下拉列表里选择“一个列值”，然后在后面选择对应列的名字。

查询更新字段

在“字段”标签页下，可以指定流里字段和表里字段的映射关系。有一点像“插入/更新”步骤里的“更新字段”部分和“数据库查询”步骤里的“查询表返回的值”部分。“Load dim_customer SCD”步骤里该标签页的配置如图8-15所示。

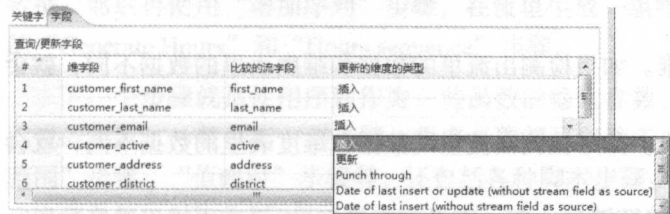


图8-15 “字段”标签页的配置

无论是在“查询”还是“更新”模式下，“维字段”里的数据，都会从维度表里取出来。

只有在“更新”模式下，“比较的流字段”里的数据才会从数据流中取出来，并和“维字段”相比较，以判断是否是发生变化的维度。

如果比较“维字段”和“比较的流字段”后发现不一样，那么要从“更新的维度的类型”里选择一种操作方式。可以为每个列选择不同的更新方式，例如有些列可以按照类型1的缓慢变更维管理，有些列可以按照类型2的缓慢变更维管理。

对于类型2的缓慢变更维来说，可以选择“插入”选项，当有变化时就会插入一条数据。

8.3.4 其他类型的缓慢变更维

前面讨论的类型1和类型2的缓慢变更维是最常使用的，但也存在其他一些缓慢变更维。实际上，很多情况下，一个维度并不是一个纯粹的类型1或类型2的缓慢变更维。要根据属性选择不同的策略。

类型3缓慢变更维

Kettle没有一个专用的步骤来支持类型3缓慢变更维。但不用花太多时间，你就可以实现这一类的缓慢变更维。

例如，若你只对维度数据里的当前值和上一个值感兴趣，可以使用“数据库查询”步骤来查询当前的值，并把这个值保存成上一个值，然后使用“更新”步骤把流里的数据保存成当前值。如果想动态地增加列来维护不同版本的值，可以写一个作业并使用“表里的列是否存在”步骤来判断是否要更改表结构，然后使用SQL脚本步骤，执行相应的DDL语句增加一个新列。

混合缓慢变更维

前面说过，缓慢变更维度的类型可以不局限在表级，可以是列一级，不同的列可以使用不同的缓慢变更维。一个维度表如果同时使用了不同类型的缓慢变更维，就是一种混合的缓慢变更维。

当要决定为某个列选择哪种类型的变更方法时，并不能由列本身来决定，而应是用户是不是关注这个列的变化历史，这是给列选择哪种类型的变更方法时要考虑的问题。

例如用户的生日字段。如果发现源系统生日字段发生了变化：这种情况只能解释为前面的生日是错误的，数据的变化是对前面数据的修正。但是如果是客户的名字发生了变化，可能是拼写错误，也可能是客户结婚了，使用了配偶的名字。

在“维度查询/更新”步骤里可以为每一列设置不同的维度更新类型，可以选择的更新类型如下。

- **插入**：这是类型2的缓慢变更维。如果检测出流里的数据和维度表里的数据不同，就会往维度表里插入一行。
- **更新**：这是类型1的缓慢变更维。如果检测出流里的数据和维度表里的数据不同，就会直接替换原来的数据。
- **穿透**：也是更新。但不只更新匹配上的一行，而是更新类型2缓慢变更维里维度数据行的所有版本。
- **上次插入或更新的日期（没有流字段作为数据源）**：让步骤自动维护一个日期字段，把系统日期记录下来，作为最后插入或更新的日期。
- **上次插入的日期（没有流字段作为数据源）**：让步骤自动维护一个日期字段，把系统日期记录下来，作为最后插入的日期。
- **上次更新的日期（没有流字段作为数据源）**：让步骤自动维护一个日期字段，把系统日期记录下来，作为最后更新的日期。
- **最新的版本（没有流字段作为数据源）**：让步骤自动维护一个标记字段，来标记当前数据行是不是最新的版本。

8.4 更多维度

下面看一些其他类型的维度。

8.4.1 生成维（Generated Dimensions）

对于某种类型维度来说，如日期和时间维度，这些维度可以提前生成。其他生成维的例子还包括一些小型维度，如人口统计维度等。

日期和时间维度

在第4章里，我们描述了转换 `load_dim_date.ktr`（图4-4）和 `load_dim_time.ktr`（图4-5），这两个转换用来加载sakila租赁星型模型里的 `dim_date` 和 `dim_time` 维度表。这些维度的构造前面已经详细讲过，这里不再重复。

关于如何生成本地日期的维度表，还可以参考网址：<http://rpbouman.blogspot.com/2007/04/kettle-tip-using-java-localesfor-date.html>。

Kettle的 `samples/transformations` 目录下也带了几个例子，用来生成日期维度表：

- General - Populate date dimension AU.ktr

■ General - Populate date dimension.ktr

小生成维

在很多情况下，如人口统计维度这样的小型维度也可以通过提前生成的方式获得。一般生成这样维度的转换和第4章的图4-5的转换非常相似。

这种转换一般都是用几个“生成记录”步骤，每个步骤用于生成维度表里某个独立类型的数据。然后再使用“增加序列”步骤，在流里生成一组唯一的标识。参考load_dim_time转换里的“Generate Hours”和“Hours sequence”步骤。

下一个步骤就是使用序列作为一些函数的输入参数，最后生成一些描述性的数据。这里我们说的函数，一般是指一些步骤。很多步骤都可以用于这种用途，如“计算器”步骤、“数值范围”步骤、“值映射”步骤等。还包括各种脚本步骤，如“Java脚本”步骤、“公式”、“用户自定义的Java表达式”等。参考load_dim_time转换里的“Calculate hours12”步骤。

再下一个步骤就是要使用“记录关联（笛卡儿输出）”步骤，把这些独立生成的数据流连接起来。最后形成包含所有数据的一个数据流。这个数据流还需要生成一个键。可以使用维度表的属性列生成一个智能键，如load_dim_time转换里的“Calculate Time”步骤，也可以直接使用“生成序列”步骤生成代理键，或者利用数据库的auto_increment或IDENTITY这样的自增列属性自动生成代理键。最后使用“表输出”步骤输出到维度表中。

说明：Pentaho Solutions一书的第10章里的dim_demography.ktr转换就是生成小维度表的例子。我们从那本书的网站上下载这个例子<http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470484322.html>。我们也可以从下面的图8-16看到这个例子。

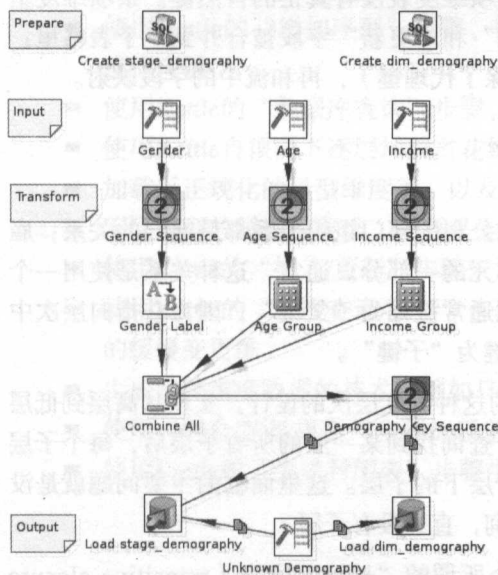


图8-16 生成一个人口统计的小维度

8.4.2 杂项维度（Junk Dimensions）

很多小维度都是一些杂项维度：它们包含了很多不同的和不相关的属性，这些属性对分析

会有一些帮助,但还不能把这些维度分类。通常,杂项维度是从一个大维度中提取出来的。如客户属性维度里可能包含了客户的自然属性(人口统计方面),还包含了他们的付款行为和购物行为方面的属性。

像人口统计这类的小维度通常可以提前生成,但有些小维度并不能提前生成。可能是因为小杂项维度的组合非常多,或者小杂项维度的取值不能提前确定。

“联合查询/更新”步骤适合于实现这类杂项维度。在Spoon里,“联合查询/更新”步骤位于左侧步骤树的“数据仓库”类别下。

说明: 在第4章的load_dim_film转换里,在加载dim_film时曾看到过“联合查询/更新”步骤(图4-16)。但dim_film维度表不是一个杂项维度,所以我们没有在那里详细介绍“联合查询/更新”步骤。

打开“联合查询/更新”步骤的配置窗口,首先看到的一些选项和“数据库查询”、“插入/更新”、“维度查询/更新”步骤里的选项都类似:

- 数据库连接、表名、模式名的输入框。
- 维度表里的列和数据流里的字段映射的输入。
- 指定维度表里的代理键和控制如何生成代理键的选项。

“联合查询/更新”步骤和“插入/更新”步骤非常类似:它利用关键字去维度表查找对应的数据,如果能找到,就更新这条记录;如果没有找到,增加一条记录。记录的代理键会被追加到步骤的输出流里,在“联合查询/更新”步骤后面的其他步骤里就可以使用这个代理键。

“联合查询/更新”步骤和“插入/更新”步骤的主要区别就是前者没有区分键字段和查询字段:杂项维度的特点就是不相关属性的所有组合,杂项维度表没有真正的自然键。杂项维度里的自然键实际就是各个属性列的组合。所以,“查询”和“更新”字段被合并到一个表格里:一般在关键字段表格里输入杂项维度表里的所有列(除了代理键),再和流中的字段映射。

8.4.3 递归层次

一些层次是递归的。如雇员和上级之间的关系或公司的部门组织关系都是递归的关系:雇员的上级本身也是雇员,部门本身也是更高层组织单元的一部分。通常,这种关系是使用一个指向本表的外键来实现(所谓的邻接表)。外键列也通常被称为“父键”,因为它指向层次中当前级别的上一级记录。类似的,我们称当前记录的键为“子键”。

类似Mondrian这样的ROLAP服务可以使用上面的这种维度层次的设计,实现从高层到低层的钻取。但是,它们一次只能钻取一层:当使用一个查询找到某一层的所有子层后,每个子层还有子层,还需要对每个子层使用另一个查询找到子层下的子层。这里面临的主要问题就是没有方便的方法去计算所有的子层聚集,必须要逐层查询,直到没有子层。

这里有一个解决上述问题的方法:可以定义一个所谓的“过渡封闭表(transitive closure table)”或简称为“封闭表(closure table)”。图8-17是一个简单的实体关系图,里面有递归关系的雇员表和它对应的封闭表。

封闭表中至少有两列来源于递归表:一列是递归表中的原始键,另一列是指向本表的外键。从递归表的顶层开始查找所有的子层,把找到的各子层的主键都放到封闭表里。最终把各

层之间全部祖先和后代关系的集合都保存到封闭表，这样将来ROLAP的SQL查询可以聚集某一层下所有子层的数据。

Kettle提供了“生成封闭表（Closure Generator）”步骤，来从递归表生成封闭表。这个步骤在“转换”类别下，这个步骤的配置如图8-18。

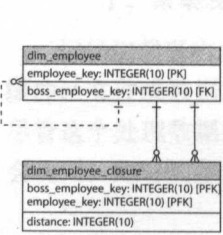


图8-17 递归表和它的封闭表

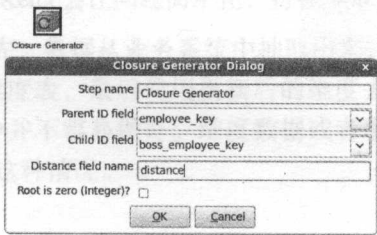


图8-18 “生成封闭表”步骤的配置

“生成封闭表”步骤就是把输入流的数据生成封闭记录，并放到输出流中。只要指定哪个字段是“父键”字段，哪个字段是“子键”字段，就可以配置好。另外还要指定一个“距离”字段，距离是祖先和后代之间的层次距离，该步骤会自动计算并发送到输出流中。

8.5 小结

本章讨论了Kettle如何管理数据仓库的维度表。主要包括：

- Kimball 的ETL子系统的理论框架如何应用到实践中，来完成维度表的加载和管理。
- 业务键和代理键，它们在数据仓库中的作用。
- 使用Kettle的“增加序列”步骤、数据库序列、auto_increment或IDENTITY 等自增列生成代理键。
- 使用Kettle的“数据库查询”步骤，把正规化的表连接起来，生成反正规化的维度表。
- 使用Kettle自顶向下逐层加载雪花维度。
- 加载反正规化的星型维度表，以及复杂的星型维度表的增量数据捕获。
- 不同类型的缓慢变更维，主要是类型1和类型2。
- 使用Kettle的“插入/更新”步骤来加载类型1的缓慢变更维。
- 使用Kettle的“维度查询/更新”步骤来加载类型1和类型2的缓慢变更维，以及查询类型2的缓慢变更维。
- 生成特殊维度数据的技术，例如日期和时间维度、小维度。
- 使用“联合查询/更新”步骤加载杂项维度。
- 使用Kettle的“生成封闭表”步骤生成“封闭表”，来扁平化递归结构的表。

第9章 加载事实表

事实表是用于分析的详细的业务数据的集合，它数据量大，会消耗大量的存储。在一个典型的数据仓库环境中，事实表会占用大部分存储空间，事实表的大小可以到几个G、几个T，甚至几个P的数据。为了把数据从源系统中加载到数据仓库里，需要一个快速加载机制，本章介绍其中的几个快速加载组件。首先我们讨论批量加载，Kettle提供了不同插件以利用不同数据库的批量加载能力。

说明：尽管本章的主题是加载事实表，Kettle批量加载的功能也可以用于其他批量加载工作，例如把文本文件加载到数据缓冲区。

在把数据加载到最终的目的表之前，还要做一些其他操作，例如查询正确的维度代理键，在本章也会介绍这方面的技术，在第19章我们介绍另一种单纯使用SQL的技术。

在本章的最后，我们介绍Ralph Kimball提出的三种不同的事实表。我们另外还介绍第四种事实表，由荷兰数学和数据建模专家Harm van der Lek博士提出的“面向状态的事实表（state-oriented fact table）”。本章最后介绍聚集表的使用，以及如何使用Kettle处理聚集表。

开始之前，我们先看看ETL 34种子系统里哪些是关于事实表的。实际本章和前面的一章共同组成了ETL子系统里的“数据发布”部分。本章覆盖了“数据发布”部分的下面几个子系统。

- 事实表加载子系统（子系统13），用来把数据加载到不同类型的事实表中。
- 代理键管道（子系统14），是多维数据仓库中的一个重要概念，本章将深入介绍，包括Kettle进行维度主键查询的不同机制。
- 多值维度桥接表生成系统（子系统15），用来构造所谓的桥接表。第4章中有这类表的例子，用来连接电影表和演员表。
- 迟到数据处理（子系统16），涵盖了如何加载迟到的维度数据和迟到的事实数据。迟到的事实数据并不是一个大问题，因为很容易可以查询到实际交易时的维度键。而早到的

事实数据（迟到的维度数据）是一件头痛的事情，我们讨论几种可能的解决方法。

■ **聚集构建（子系统19）**，用来预先统计汇总数据，提高分析性能。目前，没有一种开源数据库提供类似Oracle或其他商业数据库那样的基于物化视图的自动聚集导航功能。但我们通过Mondrian仍可以利用聚集表来提高查询性能。聚集表的难点就在于事实数据发生了变化了，聚集表里的数据也要相应变化。使用Kettle会让问题简单化，将在本章后面演示。

记住加载事实表是ETL过程的最后一步。首先，数据从业务系统中抽取出来，并写入到一个缓冲区文件或数据库表。然后，要更新所有的维度表，最后使用更新后的维度去加载事实表。尽管这个处理数据的过程看上去简单，但实际中并不容易做好。维度数据或者事实数据都可能会晚到，此时就需要迟到数据处理子系统来处理这种情况。

9.1 批量加载

如果数据量不大，几千行甚至几十万行，使用标准的DML（Data Manipulation Language）语句就可以很好地把数据加载到数据库表中。

但如果要加载几百万行甚至几十亿行数据，就不能再使用Insert 这样的语句了。为什么？很简单：因为所有的DML语句，如insert, update和delete操作，都被数据系统日志记录下来。也就是说除了要往表里插入数据，这些数据还被写入到数据库的事务日志中。另外，所有数据库列的约束还要检查数据。日志和约束检查会严重影响性能，所以从性能优化的角度，就要使用“批量加载”的处理方式。基本上，有如下两种批量加载的方法。

- **基于文件**：数据在文本文件里，要从磁盘读取。通常数据库需要一个批量加载定义文件（控制文件 control file），来确定分隔符、要加载的字段等信息。基于文本的方式适用于下面的两种场景：第一种场景，文件是已有的，而且可以直接被加载到数据库里；第二种场景，要加载的数据来源于一个转换。在第二种场景下要把转换里的数据生成文件，这种方法有优点也有缺点。主要的缺点是需要生成一个中间文件，然后才能继续后面的批量加载过程，这增加了两次额外的I/O开销。优点也很明显：产生的中间文件增强了解决方法的可靠性。如果批量加载失败，可以直接再使用中间文件加载。有时，能直接用来加载的文件是由其他系统生成的，不能直接访问。
- **基于API**：不用再存储到中间文件。使用数据库提供的批量加载API或STDIN，把数据流里的数据直接加载到数据库。理论上，这样更容易而且更快，因为这个过程没有太多的磁盘I/O，而且不用再指定和创建控制文件。但是Kettle目前没有能调用Oracle 或 Microsoft SQL Server批量加载API的插件。

每一种数据库都有自己批量加载的方法，各自的前提和方法非常不同，有最简单的MySQL里使用的LOAD DATA INFILE 文件，也有复杂的Oracle 里使用的SQL*Loader 命令。但不是所有的数据库都可以把其他外部程序的输出直接作为批量加载的输入。为了能利用Oracle SQL*Loader的直接路径加载（Direct Path Loading）功能，外部程序必须要调用Oracle Call Interface（OCI）。

9.1.1 STDIN和FIFO

一些数据库，如PostgreSQL和MonetDB可以把 STDIN作为批量加载的源。STDIN是标准输

入 (standard input) 的缩写, 可以直接在程序中作为数据流使用。例如, 你在键盘上输入字符, 字处理软件就把你的输入作为标准输入。类似的, Kettle的输出也可以作为标准输入, 提供给批量加载过程。Kettle使用FIFO 做数据的传送。

FIFO (first in, first out) 是先进先出的缩写, 就是说数据进来的顺序和出去的顺序相同。FIFO 的另一个名字是命名管道 (named pipe)。对于大多数熟悉Linux的用户来说, 管道是一个非常熟悉的概念。例如, 当执行命令 `ls -l | grep 06` 时, 会先执行 `ls -l` (所有文件名列表) 命令, 再通过管道 (`|`) 把结果传给 `grep 06` 命令 (只显示包含字符06的输出)。这两个命令之间交换的数据都是保存在内存中, 在命令之间传递, 通过其他方法无法访问到。如果使用命名管道来交换数据, 情况就不一样了。命名管道是一种特殊类型的文件, 也可以通过 `ls` 命令查看, 但实际上它存在于内存中。通过 `mkfifo` 命令可以创建命名管道, 然后就可以使用 `fifo` 来发送和接收数据。下面看一个简单的例子, 打开两个终端屏幕, 在一个终端中输入下面的命令:

```
mkfifo mypipe
ls-l > mypipe
```

然后到另一个终端, 输入下面的命令:

```
cat< mypipe
```

可以发现, 在第一个终端输入完命令行后, 没有任何反应, 直到第二个终端输入命令后, 第一个终端的命令才执行。这就是命名管道的作用, 因为只有命名管道的两端都连接后, 数据才会开始流动。如果使用相反的顺序 (先 `cat < mypipe`, 再 `ls -l > mypipe`), 会发现结果还是和刚才一样。所以管道先连接哪一端没有影响。

我们这么详细解释命名管道是因为MySQL的批量加载步骤就基于命名管道。很多使用开源数据库的人都在使用MySQL。

9.1.2 Kettle批量加载

前面讲过, 每种数据库都有自己批量加载的方法。就是说, 对每种数据库都要有一个批量加载的步骤。Kettle对大部分数据库都提供了批量加载的步骤, 例如Oracle、SQL Server、MySQL和PostgreSQL等, 也包括一些不常见的数据库, 如Greenplum、MonetDB和LucidDB等。本章不详细介绍每一个步骤, 仅快速简要介绍这些步骤, 另外, 我们提供了一些参考资料的地址, 在参考资料里有更详细的功能说明。你可能会发现没有DB2数据库的批量加载步骤, 如果想使用DB2的批量加载, 只能把数据导出成一个CSV文件, 再用DB2自己的加载工具去加载, 或者自己写一个插件。

你可能也注意到作业和转换里都有批量加载的组件。这样做也是有原因的: 所有文件管理操作都在作业里, 作业是有顺序的。作业下面的批量加载作业项都需要先把文件准备好, 而转换里的批量加载可以直接从数据流里读取数据。下面的章节详细介绍转换里的批量加载。

MySQL批量加载

MySQL可能是使用最广泛的数据库, 但它不一定是数据仓库的最好的方案。Kettle里MySQL批量加载功能很丰富。MySQL是Kettle支持的唯一能从数据库批量加载到文件的数据库, 这实际是批量抽取而不是批量加载。另外还有两个组件提供批量加载到数据库的功能: 一

一个是作业里的作业项，把文本文件批量加载到数据库里，另一个是转换里的批量加载步骤。我们前面已经解释了区别。作业项需要提前准备一个要加载的文件，而转换直接使用了FIFO把数据流里的数据加载到数据库。可以从下面的资源找到更多的MySQL批量加载的功能描述：<http://dev.mysql.com/doc/refman/5.1/en/load-data.html>。

LucidDB批量加载

Kettle的前一个版本提供了一个基于FIFO的LucidDB流加载步骤，但自从Kettle4.0起，有一个新的步骤可以直接把Kettle的数据写入到LucidDB列数据库中，这个步骤代替了上个版本的步骤。这个步骤的完全使用说明和例子可以参考<http://pub.eigenbase.org/wiki/LucidDbPDISstreamingLoader>。我们这里不再赘述，仅简单说明几个需要注意的问题：

- LucidDB的批量加载比大多数的批量加载都要智能；相对于一般的插入式的批量加载，LucidDB批量加载是合并/更新的模式。
- 除了Kettle、LucidDB和Pentaho里的其他组件也搭配得很好。例如LucidDB和Mondrian的结合就非常优秀。
- 作为列存储的数据仓库，LucidDB提供了比MySQL和PostgreSQL这些行存储数据库高十倍的性能。

关于LucidDB更多的内容可参考<http://luciddb.org/>。

Oracle批量加载

Oracle批量加载是Kettle里一个令人眼花缭乱的组件，因为这里有非常多的选项可以设置，还要设置不同种类的文件。Oracle的SQL*Loader（批量加载工具）已经有几十年的历史，但仍旧是目前使用非常广泛的一款数据加载工具。如果你没用过SQL*Loader，需要先参考Wiki：<http://wiki.pentaho.com/display/EAI/Oracle+Bulk+Loader>。关于SQL*Loader更多的内容，你还可以参考Oracle的文档<http://www.oracle.com/pls/db111/homepage>并搜索“SQL*LoaderConcepts”。

关于Oracle批量加载也有好坏两个方面。坏的一方面是：需要非常多的选项，在开始加载数据之前，需要很多准备和配置工作。好的一方面是：这款工具非常健壮，可以精确控制如何处理数据，以及可能的错误数据。Kettle也可以帮助你创建控制文件，根据输入数据流的元数据在运行中创建。

PostgreSQL批量加载

PostgreSQL批量加载还在试验阶段，它可以把Kettle的数据直接通过PostgreSQL标准的COPY命令加载到数据库里。PostgreSQL手册里的COPY命令如下，包括一些参数，用来指定分隔符、逃逸符等：

```
COPY tablename [ ( column [, ...] ) ]  
FROM { 'filename' | STDIN }
```

可以看到，文件名和标准输入都可以作为输入选项，但是Kettle只使用了后者。关于这个步骤更详细的内容请参考<http://wiki.pentaho.com/display/EAI/PostgreSQL+Bulk+Loader>。

说明: 为了让 PostgreSQL 批量加载步骤可以工作, 需要先定义一个到服务器的信任连接, 具体参考<http://wiki.pentaho.com/display/EAI/PostgreSQL+Bulk+Loader>。

表输出

尽管表输出不属于批量加载步骤, 但在前面说过, 当加载几千行或更多行数据时, 使用表输出也是一个很好的选择。在第15、16章可以看到, 如果在步骤上同时执行多份拷贝, 也会提高吞吐量。使用表输出步骤也有一些好处, 如易于使用、有字段映射设置、适用于各种数据库等。

9.1.3 批量加载一般要考虑的问题

当使用批量加载时, 尤其是要用到中间文件时, 还要考虑一些问题。要确保文件系统有足够的空间, 另外前面也讲过, 额外的文件I/O还会影响性能。幸运的是, 随着内存和存储技术的进步, 以及设备价格的不断下降, 这些都可以帮助你解决性能瓶颈问题。当遇到性能问题时, 考虑下面的几种技术:

- 使用SSD 设备作为批量加载时读写文件的存储设备。
- 如果有足够内存, 创建RAM磁盘。
- 使用第16章描述的并行和集群技术。

但无论如何, 都会存在性能和可靠性的平衡问题; RAM 上的数据不是物理保存的, 最快的RAID 选项是RAID 0, 也是最不安全的RAID 选项。

另一件要注意的事情是: 一般都有两种类型的批量加载, Insert和Truncate。Insert 操作是追加数据行, 表里原有的数据不受影响。Truncate操作先把表里的数据都清空。可以使用这些操作来优化你的方案。一般来说, 对于缓存区里的表, 应该在加载前做Truncate 操作, 对于数据仓库里的事实表, 一般都用Insert操作。但加载数据仓库里的事实表, 也有几个策略, 尤其当事实表非常大时。这些策略取决于事实表的类型。本章后面将介绍不同类型的事实表, 以及每种事实表的不同加载策略。

9.2 维度查询

加载事实表时, 要完成的一个重要操作就是要查询到维度表里正确的代理键。在前面章节里我们介绍了如何生成代理键, 并解释了为什么需要代理键, 尤其在保存历史数据 (SCD 2类型) 时更需要代理键。使用这种单列整数代理键的另一个原因是节省空间, 相对于源系统的业务主键, 在事实表里如果只引用维度表的代理键, 会节省大量的空间。当存储几亿行数据的时候, 每一行节省一个字节, 就会节省很多存储空间, 查询效率也会提高。

9.2.1 维护参照完整性

对于一个多维数据仓库来说, 事实表里的每个外键都对应维度表里的一个主键。在传统事务数据库中, 一般是通过外键约束来强制执行这种参照完整性。这种强制约束可以防止下面问题的产生:

- 删除了维度记录，但和维度相关联的事实数据还存在，引用着不存在的维度键。
- 事实表里插入一条新的数据，但它的外键所对应的维度表的主键并不存在。

但是在数据仓库里使用外键约束并不一定可行，尤其在加载大量数据的时候。数据库必须逐条检查插入的数据，降低了加载速度。在数据仓库里，不使用外键没有太大问题，但是：不能删除维度记录，这样已加载的事实表里总能找到对应的维度。另外在加载事实表时，都要查询维度表的代理键，这样新加载的数据也有对应的维度。至少理论如此，对大多数人来说实际也是如此。

关于外键约束还有几个方法：可以使用外键约束，但是只能在事实表加载完之后再使用外键。就是说在加载开始前使用“执行SQL脚本”步骤删除所有的外键约束，然后再加载事实表，加载完之后再使用“执行SQL脚本”步骤创建所有的外键约束。如果创建外键约束的过程发生错误，可以立刻知道数据仓库违背了参照完整性，并采取相应措施。

与数据仓库和ETL里的其他问题一样，很难说哪个方法是最好的。在99%的情况下，使用一个设计良好的代理键管道再加上合适的CDC技术就可以很好地工作了。再结合第11章要讲的测试技术，就可以确保正确加载数据仓库而不用再做额外的开发工作，也可以节省删除和重新创建外键约束需要的时间。当然也有一些理由使用外键约束。我们前面提到了一个（检测事实表的加载错误），另外还有一个是性能问题；大多数数据库会使用外键来优化查询执行计划，以达到更好的查询性能。

下面部分介绍查询代理键所用到的一些技术。

9.2.2 代理键管道

找到正确的维度代理键的方法非常简单，这种方法形象地被称为代理键管道，如图9-1的简单例子，例子里用了三个数据库查询步骤。为了简化问题，后面使用了一个“表输出”步骤来代替某个批量加载步骤。



图9-1 代理键管道

第4章曾经介绍了一个更详细的例子，根据sakila数据库里的租赁日期抽取维度主键。可以把上面的查询过程翻译成通用的伪SQL语句。

```
SELECT      dimkey
FROM        dimtable
WHERE       fact_businesskey = dimtable.businesskey
AND         valid_from <= fact_date
AND         valid_to   >  fact_date
```

这个 SQL 表示抽取业务键相等，而且业务发生时间在维度有效期内的维度的主键。因为要比较多个条件，所以查询维度主键的速度不能再快了。如果每次加载过程都是稳定的，没有晚到的维度数据或事实数据，也可以直接抽取维度表里相同业务主键的最后一记录，这样可以加快查询速度。

使用内存查询

最快的查询方式就是从计算机的RAM中查找。所以最快的找到某个代理键的方法就是把所有维度保存在内存中，然后在内存中查找。“数据库查询”步骤中的“使用缓存”和“从表中加载所有的数据”选项可以实现在内存中查找。但要小心这种方法容易造成内存溢出，尤其是一个转换里有多个数据库查询步骤。另外要限制列数，只使用真正需要的列来做查询。如果你需要的只是业务键和代理键，以及可选的开始/结束日期列，那么就没有必要把维度表全部都加载到内存里。

提示：最大可用内存数可以在脚本里（Spoon、Carte、Kitchen和Pan）通过-Xmx 参数设置。如果在Windows下使用Kettle.exe 命令，在Kettle ini文件里设置 kettle.14j.ini。

流查询

在大多数情况下，“数据库查询”步骤可以很好地工作。但它不能解决所有问题。如果要查询的数据来源于一个外部数据源，或者不在数据库里，该怎么办？这时需要使用“流查询”步骤。这个步骤没有“数据库查询”步骤那么多的选项，它只允许做等值查询。但这个步骤支持从各种数据源和其他步骤查询数据。在一些场景下，如维度数据和事实数据能同时准备好，先使用“表输入”步骤获取每个业务键最后一个版本的维度记录，然后再使用“流查询”步骤把“表输入”步骤的结果作为输入，是查询大型维度表的最快查询方式。

上面的方法速度快，是因为查询集里只包括了实际需要的记录，若客户维度包括了三百万行记录（包括了历史记录），当前最新的版本的数据可能只有总数的1/3（这是很普遍的情况），所以只要用流查询步骤在一百万行数据中查找就可以。即使是同样多的数据行，使用流查询步骤也快一些，图9-2和图9-3说明了这种情况。我们使用1GB的TPC-H测试数据（关于TPC-H测试数据参考www.tpc.org）。1GB的数据有150 000 行客户数据，1 500 000行订单数据，大约6 000 000行的详单数据。例子里使用的客户维度表数据来源于TPC-H的这些客户数据，只是通过“增加序列”步骤增加了一列 customer_id。

图9-2展示的流查询的过程，查询数据来源于“表输入”步骤，表输入步骤里使用下面的SQL 语句来获取维度数据：

```
select customer_id, c_custkey from dim_customer where current_flag = 1
```

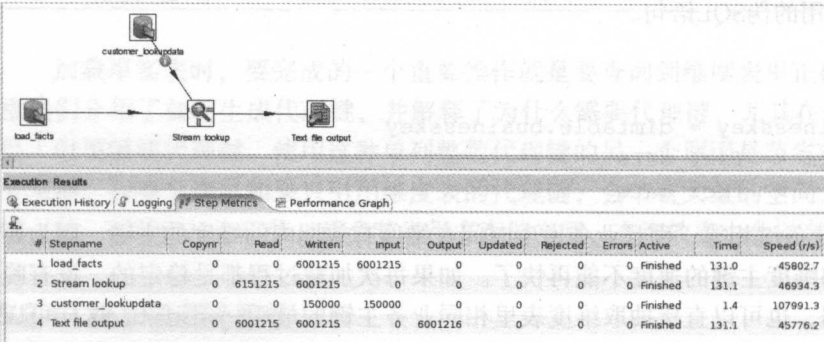


图9-2 使用流查询步骤

说明：图9-2的例子使用了一个“表输入”步骤作为“流查询”步骤的数据来源。实际上任何步骤都可以作为流查询步骤的数据来源。

流查询步骤的配置如图9-3所示，配置的过程也很简单，先要设置一个数据来源步骤，然后再设置一个查询键。

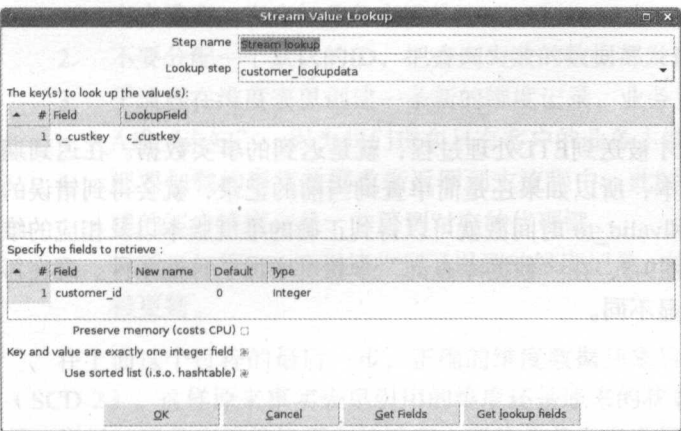


图9-3 流查询步骤的配置

查询键可以是多列，但必须是等值查询。在这个例子里，使用订单表里的客户键（o_custkey）去查找客户维度表里的业务主键（c_custkey），最后返回的字段是customer_id。不要忘了指定字段类型；默认是没有选项（-），如果没有选择，后面的步骤会发生错误。底部的三个选项，用于根据不同的情况调整Kettle压缩和排序数据的过程。

说明：关于“流查询”步骤内存选项的说明和讨论参考下面的网址：<http://forums.pentaho.org/showthread.php?t=68058>。

作为对比，我们也可以使用“数据库查询”步骤运行同样的转换，结果如图9-4所示。

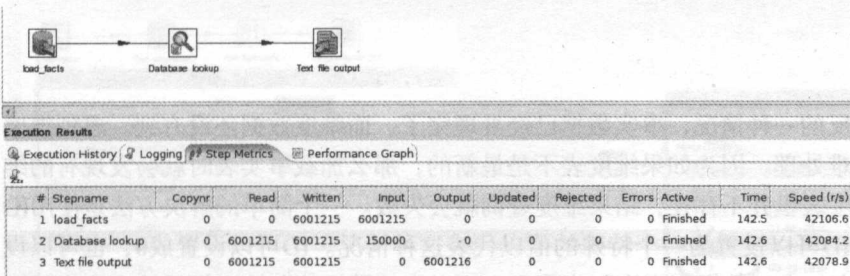


图9-4 “数据库查询”步骤

仔细观察一下，可以发现在数据相同的情况下，流查询步骤比数据库查询步骤快一些。但很难说是否应该优先使用“流查询”步骤，从例子可以看到，使用“流查询”步骤至少要多使用一个步骤，而且流查询步骤只提供简单的等值比较。在很多场合下可能需要更多的查询条件，例如，当处理迟到的事实表数据时，就需要“数据库查询”步骤。在下面的章节中我们就介绍如何处理迟到的数据。

9.2.3 迟到数据

在正常情况下，Kettle是有规律地从源系统中抽取和处理数据。在本章的介绍部分曾经说过，标准的顺序应该是先处理维度表再处理事实表。这个顺序很重要，因为我们需要事实表里的指向维度表的代理键，而代理键需要通过维度加载过程生成。但如果数据没有顺序到来，例

如，事实数据或维度数据晚了几天甚至几个星期才到来怎么办？这样加载的顺序就会被打乱；这样可能是有的客户没有订单，或有些订单指向了一个不存在的客户。下面的章节说明如何处理这类问题。

迟到事实数据

当交易发生很久以后，交易数据才被送到ETL处理过程，就是迟到的事实数据。在迟到期间，相关的维度数据可能有了新的版本，所以如果还是简单查询当前的记录，就会得到错误的维度代理键。此时要使用 valid_from和valid_to 时间戳就可以得到正确的维度版本以及相应的维度代理键。但容易并不代表快速。看图9-5，在“数据库查询”步骤里增加了日期条件。现在再和图9-4比较，我们就会发现速度上明显不同。

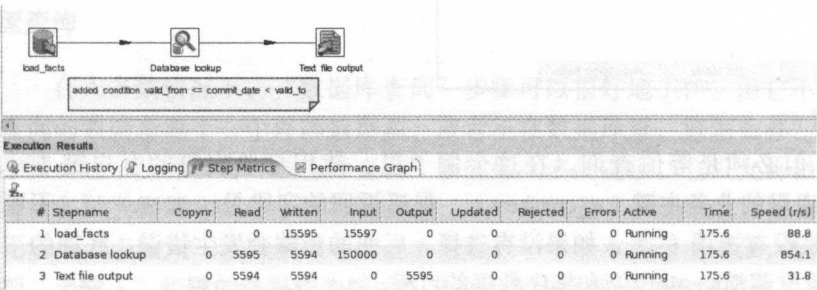


图9-5 加载迟到的事实数据的性能

这里要说明的是，如果有迟到的事实数据，因为要使用valid_from和valid_to字段来比较查找维度代理键，加载事实数据总体性能会比较低（即使使用了索引），所以最好把迟到的事实数据的加载流程从主流程里分离出来。

迟到维度数据

迟到维度数据是相反的一种情况，事实数据已经处理完了，而维度数据还没有到。迟到维度数据比迟到事实数据更难处理。因为如果维度表不是最新的，那么加载事实表时就会发现有的维度，如客户，在客户维度表里并不存在，结果维度查询就会失败。一个简单的解决方法是使用ID指向一个未知的维度，ID可以设置为一个特殊的值以代表这种情况。ID可以设置成0，也可以设置成-999或其他特殊数字。在“数据库查询”步骤里，可以在“默认值”输入框里输入这个特殊ID，如图9-6所示。如果在维度表里没有找到这个对应维度的代理键，就使用这个默认的ID。

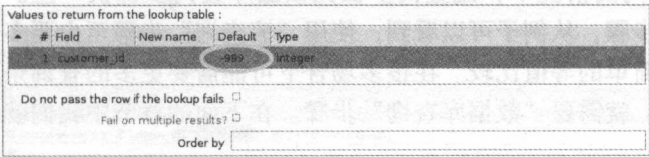


图9-6 设置默认返回值

这时如果生成报表，某个维度里会出现一个未知的分类，可能是某个未知的客户或未知的产品。

这样能解决问题吗？当然不能。另外也不可能再把事实表里的维度代理键替换为正确的代理键。所以我们要换一个角度思考，应该把“未知”，想成“还不知道”，有了这个思路我们

就能找到新的方法。下面就是我们能想到的新方法的工作流程以及如何用Kettle实现：

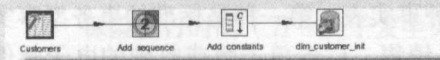
- 1. 当要加载数据到事实表时，在查找某个维度的代理键时查询失败。假设查询失败的是客户维度，客户的业务主键是ABC123。
- 2. 不要分配一个默认ID，把查询失败的数据都发送到一个子流程里。
- 3. 子流程在维度表里创建一条新的维度记录，业务主键是ABC123，其他字段都设置为N/A或“未知”，因为我们现在只有客户的业务主键。
- 4. 把要加载的事实数据重新返回到主流程中，此时再查找客户维度时，就可以找到刚创建的客户维度记录，并等到对应的代理键。
- 5. 以后，如果ETL流程接收到了迟到的维度记录，维度记录里“未知”数据的字段会自动被更新。

在上面这个过程的最后一步，正确的维度数据到来后，可能会在维度表里又插入一条记录（SCD 2），这样原来事实表里引用的维度还是原来的状态，所有的字段还是N/A或“未知”。是这样做，还是直接替换原来的维度，要从业务上来选择。保留原来的维度，可以把这种错误情况记录下来。但是我们也可以不插入新的维度，而直接更改原来的维度。

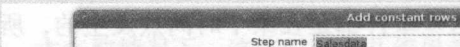
例子：使用Kettle处理迟到维度数据

在这个例子里有一个客户维度表和一个事实表，客户维度表里有几个字段，事实表里有customer_id、sale_date和sale_amount几个字段。转换开始使用“自定义常量数据”步骤生成六条客户数据，如图9-7所示。

注意表里有六条客户记录，都加载到了dim_customer 维度表里。下面我们要创建一些销售数据，在这些销售数据里有一条销售数据的客户不在 dim_customer 维度表里。图9-8是这些销售数据，注意cust_key 7 在维度表中不存在。



| Add constant rows | | | |
|---------------------|----------|-----------|--------------|
| Step name customers | | | |
| Meta Data | | | |
| # | cust_key | cust_name | cust_country |
| 1 | 1 | Doug | USA |
| 2 | 2 | Richard | USA |
| 3 | 3 | James | USA |
| 4 | 4 | Matt | BE |
| 5 | 5 | Roland | NL |
| 6 | 6 | Jos | NL |



| Add constant rows | | | |
|---------------------|----------|------------|-------------|
| Step name salesdata | | | |
| Meta Data | | | |
| # | cust_key | sale_date | sale_amount |
| 1 | 1 | 21-05-2010 | 23 |
| 2 | 1 | 30-06-2010 | 15 |
| 3 | 3 | 11-05-2009 | 76 |
| 4 | 6 | 15-02-2010 | 82 |
| 5 | 7 | 03-09-2010 | 45 |

图9-7 默认查询返回值

图9-8 销售数据的例子

这些数据将被发送到“数据库查询”步骤，来查找客户维度表的customer_id 代理键。前四条记录都可以找到对应的客户维度，但是cust_key =7 的记录找不到对应的客户维度记录。为了处理查询失败的数据，我们给“数据库查询”步骤增加一个错误处理步骤。这些数据都可以存到一个文件里、一个数据库里，甚至使用“复制记录到结果”步骤保存到临时的内存空间里。在这个例子里，我们把这些数据保存到一个文件里，如图9-9所示。

现在我们就可以把这些分离出来的客户维度加载到维度表中，在维度表中创建新的维度记录，这个过程如图9-10所示。

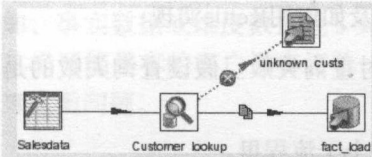


图9-9 事实数据的错误处理

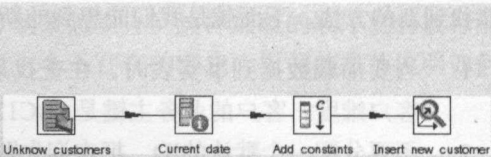


图9-10 添加客户维度

首先从文本文件中读取这些维度，然后使用一个“获取系统信息”步骤加入今天的日期。接着使用一个“增加常量”步骤，将维度中没有的字段设置成“N/A”，如图9-11所示。

转换的最后使用“维度查询更新”步骤生成新的customer_id，并增加到客户维度表中。（见前面介绍的“维度查询更新”步骤）。维度表最后的结果如图9-12所示。

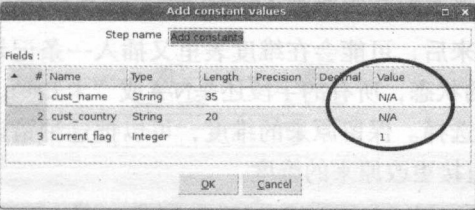


图9-11 增加N/A值

| customer_id | cust_key | cust_name | cust_country | valid_from | valid_to | cur |
|-------------|----------|-----------|--------------|---------------------|---------------------|-----|
| 7 | 7 | IWA | N/A | 2010-04-08 00:00:00 | 2999-12-31 23:59:59 | 1 |
| 6 | 6 | Jos | NL | 1900-01-01 00:00:00 | 2999-12-31 23:59:59 | 1 |
| 5 | 5 | Roland | NL | 1900-01-01 00:00:00 | 2999-12-31 23:59:59 | 1 |
| 4 | 4 | Matt | BE | 1900-01-01 00:00:00 | 2999-12-31 23:59:59 | 1 |

图9-12 增加了新的客户维度

最后再把被拒绝的事实数据重新查询一遍维度表，此时就可以查到新插入的维度记录，以及返回相应的customer_id键。

9.3 处理事实表

回忆一下第5章讲的ETL子系统，Kimball 总结了三种事实表。第一种是标准的事务型事实表，事实表里的每一行都代表了一个交易事件，如产品销售、广告单击数或收到的电子邮件数。这些事件都是在某个时间点发生的，所以每个事件都有一个时间戳。因为数据的这种特征，所以我们才能很容易地使用一些维度层次，如年、产品组或国家来汇总数据。到目前为止，本书所有的例子都基于事务型事实表。

9.3.1 周期快照和累积快照

第二种类型的事实表是周期快照表，用来定期存储某种信息的状态。在一些场合下，如果在事实表里存储每一次的交易事务会显得比较烦琐，这时可以使用快照表。例如，在企业的仓库里可能有几千种商品，这些商品经常出入仓库。为了查看商品的流动情况，把商品的每次出入库的这种交易数据都记录下来是很有意义的。但另一方面，数据仓库的管理员可能只关心每天的或每月的总的产品库存，这时只需要交易快照表就可以了。

周期快照表的例子还包括库存数量和银行账户余额或保险状态摘要等。另外还要记住，周期快照的度量是“半加法”。对于一般的事实表来说，可以按照时间、客户组、产品线等汇总销售数量。但对于周期快照事实表，不能按照所有维度汇总，尤其不能按照时间维度。周期快照表里数据本身就是一个时间周期内的汇总数据，例如，计算每个周期银行账号的余额的合计是没有意义的，应该计算最大、最小、平均的度量，这也是“半加法”这个词的含义。另外也

有“非加法”的数据，意味着不能在任何维度上汇总数据。例如，室内温度数据，每个房间每个时间点的室内温度都是不同的，可以定期获取各个房间的温度快照，但是把所有房间的温度快照求和是没有意义的，同样把每个小时的温度快照求和也没有意义。

警告：使用周期快照的时候要小心。假设你要把一个大银行几百万客户的每天的账户余额都存储到一个周期快照表里，这意味着即使客户的账户没有发生变化，每天这个客户在快照表里也会有一条数据，快照表会变得非常大。

第三种类型的事实表是累积快照表。它其实是一种特殊的事务类型事实表。这里的“累积”一词说明这个事实表会定时更新，通常在流程完成后更新。这里的“快照”一词容易和周期快照混淆。如果把这种表称为“累积事实表”可能会更明确一些，但是行业习惯上把这种表称为“快照”，所以我们也这么称呼它。这里的“快照”一词实际是指在一个过程里，处理已完成的阶段。例如一个订单事实数据里如果只有一个日期，那么这个订单是刚下单。

累积快照表的一个例子就是销售过程，销售过程包括了订单、拣货、货运、发票和付款等步骤，每一个步骤都有自己的日期。实际上，第4章的fact_rental表就是一个有租赁日期和归还日期的累积快照表。

9.3.2 面向状态的事实表

仔细分析前面介绍的三种事实表，如果你对保险数据或其他变化不频繁的数据有一定了解，你可能会发现事实表里少了一些东西。例如，给房子买火灾保险，会有下面几个时间点：

- 保险生效时间
- 每年保费金额变化的时间
- 投保总额变化时间
- 保险执行时间（万一房子发生火灾）
- 保险截止时间

基本上，这就是保险业务里的一些关键时间点。大多数情况下，这些时间点几年里都不会变化。这些数据在数据仓库里一般是通过“周期快照表”的方式保存，保存每个保险每个月或每年的状态。这样有一些浪费，而且也没反映出来变化的时间。例如，保险在这个月的20号发生了变化，但是在下个月1号才抓取快照；发生变化的时间点，20号，在事实表中丢失了。如果只在事实表中保存状态和状态在源系统中变化的时间，可能会更好，更像一个事务事实表。这就是我们要介绍的第四种事实表，称为“面向状态的事实表”。荷兰数据仓库专家 Harm van der Lek博士首先提出了这个概念。下面对“面向状态事实表”的描述材料和例子，都得到了Harm van der Lek博士的允许。

“面向状态事实表”里的一行说明一个对象在一个时间跨度内状态没有发生变化。高层对象（被引用的表）发生了变化，认为低层对象（引用其他表的表）也发生变化。如果业务场景符合下面的几个条件，就可以考虑使用“面向状态事实表”：

1. 有多个和业务相关的要分析的低层对象。
2. 对象的变化不可预测。
3. 能捕获到变化的详细数据，包括父对象的变化。

这里有一个简单的例子：底层的对象是储蓄账户，高层对象是客户维度，客户维度是SCD 2

类型的维度。如表9-1所示，有一个客户发生了变化，这个客户在2009年7月21日从New York 搬到了Boston。

表9-1 客户维度表——SCD 2 类型

| Customerversion _key | CustomerCode | From Date | To Date | City |
|----------------------|--------------|------------|------------|----------|
| 5 | CG067 | 1900-01-01 | 2009-07-21 | New York |
| 8 | CG067 | 2009-07-21 | 9999-12-31 | Boston |

表9-2是一个引用客户维度表的“面向状态事实表”的例子，事实表里有“半加法”类型的度量值，账户余额。

表9-2 面向状态的事实表

| ACCOUNTNR | From Date | To Date | Customerversion _key | BALANCE |
|-----------|------------|------------|----------------------|---------|
| 3200354 | 1900-01-01 | 2009-07-10 | 5 | \$ 100 |
| 3200354 | 2009-07-10 | 2009-07-21 | 5 | \$ 200 |
| 3200354 | 2009-07-21 | 9999-12-31 | 8 | \$ 200 |

2009年7月10日，余额从\$100 变为\$200，所以事实表里对应一条记录（第二行）。客户更换地址，也使事实表发生变化，事实表又增加了一行，指向新的客户维度代理键，账户余额一列不变。使用这种事实表可以得到任意时间点、某个城市的账户余额的总和。例如，在2009年7月底，客户CG067的账户余额应该纳入到 Boston 的城市余额中，因为在7月底，客户CG067已经搬到了Boston。

使用这种方式存储的另一个好处是可以基于“面向状态的事实表”来定义周期快照表。因为有完整的历史数据，可以比较容易地获取快照数据。首先创建一个表，包含快照的所有时间点。例如要创建每月快照表，就要有一个每月的时间表，如表9-3所示。

表9-3 月表

| MONTHNR | LASTDAY |
|---------|------------|
| 200906 | 2009-06-30 |
| 200907 | 2009-07-31 |

有了时间表，就可以创建周期快照表，周期快照表既可以是视图（见后面的介绍）也可以是物理表，创建周期快照表的SQL 语句如下：

```
SELECT Month.MonthNr
      , SOFactTable.AccountNr
      , SOFactTable.Customer_version_key
      , SOFactTable.Balance
FROM   SOFactTable
      , Month
WHERE  Month.LastDay >= SOFactTable.From_date
      AND Month.LastDay < SOFactTable.To_Date
```

这个SQL查询结果如表9-4所示。

表9-4 周期快照表视图

| MonthNr | AccountNr | Customer Version_key | Balance |
|---------|-----------|----------------------|---------|
| 200906 | 3200354 | 5 | \$ 100 |
| 200907 | 3200354 | 8 | \$ 200 |

9.3.3 加载周期快照表

加载周期快照表并不复杂，不过需要注意在准备和获取数据时的一些问题。首先，周期快照类似于在某个时间点拍一张照片，对于ETL和数据仓库来说，就是获取某个时间点数据的状态，每个周期的时间点都应该完全相同。对于很多周期类型的业务场景，很自然会使用周期快照表。如银行业务，周期性的封账是标准业务流程的一部分，在封账期间，数据不会发生变化。所以我们就不必在每个月最后一天的00:00去抽取加载数据。

这里还有一项要注意的事情：计算每个月账户余额是一个非常麻烦的过程，最好留给操作系统去做，操作系统在设计上就是去做这类的事情。如果试图在ETL 过程去做将是一件非常烦琐的工作，甚至可能会失败。

加载周期快照本身只要使用一个简单的批量加载即可，一般使用append模式。唯一要注意的是事实表里应该有一个快照的周期ID字段。

9.3.4 加载累积快照表

加载累积快照要使用update方式。在第4章看到过，可以使用“插入/更新”步骤来加载累积快照事实表，但如果数据量比较大，使用“插入/更新”步骤的时间会比较长。还有一种方法可以代替“插入/更新”步骤，尤其在数据量比较大的情况下。但是这种方法需要数据库支持分区。但如果数据库不支持分区，我们也可以使用两个表和一个基于这两个表的视图来模拟分区。

这种方案如图9-13所示。

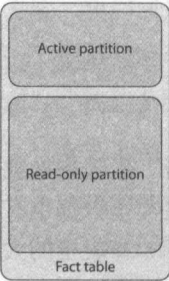


图9-13 分区累积快照表

这种方案的工作流程如下：

- 所有的插入和更新操作都发生在活动分区，活动分区相对较小。
- 累积加载完成后（所有业务流程的中间环节都有了一个时间，其他事实列也有了相应的值），把记录从当前活动分区移动到只读分区。
- 最后是一个要周期执行的工作：清理活动分区里的数据，并把这些数据从活动分区移动到只读分区，这一工作一般使用批量加载步骤把数据追加到只读分区中。

基于活动分区和只读分区的这种方案，还有一种处理流程可以完全避免“插入/更新”步骤：

- 读取活动分区中所有数据，同时truncate或delete活动分区。
- 从源系统中抽取变化的数据，和刚读取到的活动分区中的所有数据合并。
- 把完整的记录加载到只读分区，把不完整的记录加载到活动分区。

第二种方法不需要定期把数据从活动分区移动到只读分区，因为在加载过程中已经把完整的数据加载到了只读分区中。第二种方法中也不使用DML语句，这样就可以使用像InfoBright社区版这样数据库了（InfoBright社区版不支持DML）。

说明：并不是所有的数据库都支持truncate分区；有的数据库要求删除并重建分区，才能删除分区里的数据。

9.3.5 加载面向状态事实表

面向状态的事实表（SOF）是否真的是一类不同的事实表？它和维度表以及周期/累积快照事实表都比较相似。但这只是我们的第一印象，实际还有很多区别：

- SOF表包含了度量，维度表没有度量值。
- SOF表包含了指向维度表的外键，如果在一个纯星型模型里，它就是事实表，否则就是雪花模型。
- SOF表的状态事先是不知道的，而累积快照表，所有的状态都事先知道，都固定在表结构中。

但它们之间也有相似的地方：和周期快照表一样，SOF表的度量也是“半加法”（不能在所有维度上都相加）。和维度表一样，SOF表也有valid_from和valid_to日期字段。和维度表一样，SOF表中也可以增加一个current_flag字段来标识出当前活动的记录。另外从技术架构上，SOF表也应该使用代理键，以避免使用多列主键。从上面可以看出使用Kettle加载SOF表也是非常容易的，转换如图9-14所示。



图9-14 加载SOF表的例子

这个转换和其他事实表加载流程的不同之处在于，它使用了“维度查询/更新”步骤来加载SOF事实表。没有其他加载方法，因为SOF表里没有最终状态的概念。当然，有些类型数据也有最终状态，像前面说的火灾保险合同的到期或终止、银行账户关闭，等等。但是，SOF表里的大部分数据都是长期地不定期发生状态变化的数据。

9.3.6 加载聚集表

加载聚集表增加了ETL流程的复杂度和负载，但可以提高分析的性能。聚集表里包含了高层次的汇总数据。如果查询工具或OLAP引擎可以使用这些聚集表，就不用再查找事实表里的数据。例如，第4章例子里的事实表fact_rental里有16 044条记录。事实表的粒度是每一笔租赁业务，如果要统计每个客户每年的租赁情况，就要把16 044条记录都浏览一遍。当然索引能提高查询速度。但是如果有了每个客户每年租赁次数的聚集表，这表只有169条数据，要查询数据的

数量降低了100倍。

总的来说，有以下三种加载聚集表的方式。

- 使用额外步骤扩展加载事实表的转换，排序需要汇总的字段，如customer_country和year4字段，然后使用“内存分组”步骤把数据聚集，最后使用“表输出”或“批量加载”步骤写入到聚集表。
- 使用表输入步骤，表输入步骤里是下面的SQL语句，直接生成聚集表数据：

```
SELECT      c.customer_country, d.year4,
            SUM(count_rentals) as count_rentals
FROM        fact_rental f
INNER JOIN  dim_customer c ON c.customer_key = f.customer_key
INNER JOIN  dim_date      d ON d.date_key = f.rental_date_key
GROUP BY    c.customer_country, d.year4
```

- 使用“执行SQL脚本”步骤，步骤里的SQL脚本和上面的SQL类似，除了前面加上一句insert into agg_table(customer_country, year4, count_rentals（如果聚集表存在）或者create table agg_table as（如果想动态创建表，如果表已经存在需要先删除）。这些语句的语法取决于数据库。这里使用的是MySQL的语法。这种方法的优点就是完全在数据库内部执行，不用在Kettle里执行，速度比较快。

在Pentaho里，聚集表通常和Mondrian 结合使用。Mondrian 超出了本书的范围，我们的Pentaho Solutions一书里解释了如何使用 Mondrian 聚集表设计器来生成聚集表，以及如何配置Mondrian 使用聚集表。

9.4 小结

本章介绍了不同类型的事实表和使用Kettle的加载方法。首先介绍了Kettle里可用的批量加载步骤，解释了基于文件和基于API的批量加载的区别。另外还介绍了介于两者之间的命名管道的方法。我们通过简单的Linux 命令说明什么是命名管道。

加载事实表要做的准备工作之一就是查找正确的维度代理键，我们也称之为“代理键管道”。我们介绍了数据库查询和流查询，另外深入介绍了如何处理迟到的维度数据和事实数据。

本章的最后部分解释了不同类型的事实表和加载这些事实表时要注意的一些问题。下面是本章介绍的几种事实表：

- 交易事实表
- 周期快照事实表
- 累积快照事实表

接着又介绍了Harm van der Lek博士提出的一种新型的事实表，“面向状态的事实表”，并介绍了Kettle如何支持这种事实表。最后我们介绍了如何使用Kettle加载聚集表。

本章没有介绍处理大数据量事实表的性能问题。我们将在本书的第四部分介绍这一主题，并深入介绍性能调优、并行化、集群和分区等内容。

第10章 处理OLAP数据

OLAP是在线分析处理（Online Analytical Processing）的缩写，它在34个ETL子系统中占据着第20的位置。尽管处理 OLAP数据的存储只是34个ETL子系统之一，但本章的全部内容都将讲述这一主题。

OLAP这个词汇是在1993年由数据库专家 E.F.(Ted)Codd 提出的，他提出了12条规则来定义 OLAP。这些规则参见http://www.olap.com/w/index.php/Codd's_Paper。

在Codd的定义里最重要的是OLAP数据的多维特性。现在OLAP和多维这两个词几乎成为了同义词。并不是Codd发明了OLAP，他只是给它起了一个名字，从这个名字以后，围绕着 OLAP这个概念，几十亿美元的产业便出现了。在Codd提出OLAP的定义时，第一个OLAP产品已经存在了，Congnos Powerplay和Arbor Essbase可能是最广为使用的。这些产品直到今天还在使用，Powerplay是IBM-Cognos BI 产品的一部分，Essbase是Oracle BI产品的一部分。但OLAP市场上最大的厂商实际是起源于以色列的一家叫 Panorama的小软件公司。1996年，这家公司将它的OLAP技术卖给了微软，这就是微软的分析服务（Analysis Services）。现在微软的OLAP分析服务占有最大的市场份额，部署的产品数量最多。微软也为OLAP市场做了很多贡献：它创造了多维查询语言（Multi Dimensional eXpressions或MDX）。MDX现在是多维查询语言的事实标准，是做分析查询的强有力的工具。

本章将详细讲述 OLAP 技术的背景信息、配置和挑战，但不会详细讲解MDX。本章将在不使用MDX查询的情况下，讲述如何从OLAP数据库读写数据，所以即使你不熟悉MDX，也可以学习本章的内容。

说明：如果想快速学习 OLAP和MDX，请看Pentaho Solutions一书的第15章。如果要深入学习MDX查询语言，也有几本书和在线资源可以参考，我们推荐下面三本书或在线资源。

- William Pearson发表在*Database Journal*上的“MDX Essentials”系列：
http://www.databasejournal.com/features/mssql/article.php/10894_1495511_1/MDX-at-First-Glance-Introduction-to-SQL-Server-MDX-Essentials.htm。
- G.Spofford, S.Harinath, C.Webb, D.Hai Huang和F. Civardi 编写的*MDX Solution: With Microsoft SQL Server Analysis Server 2005 and Hyperion Essbase* (Wiley, 2006)。
- S. Harinath, M.Carroll, S. Meenakshisundaram, R. Zare和D. Lee 编写的*Professional Microsoft SQL Server Analysis Services 2008 with MDX* (Wrox, 2009)。

10.1 OLAP的价值和挑战

OLAP 有什么特殊的呢？通常情况下，无论数据是在文本文件里，还是在Excel里，还是在数据库里，我们都是使用行和列的方式去处理数据。在OLAP的立方体（cube）里，我们并不是用行和列去定位数据，我们使用维度（dimensions）、层次（hierachies）和单元（cells）去定位数据。为了从cube中读出数据，或把数据写到cube里，我们需要知道数据在哪里。数据的位置由相关维度的交点确定。首先我们看一个典型的cube，如图10-1所示。

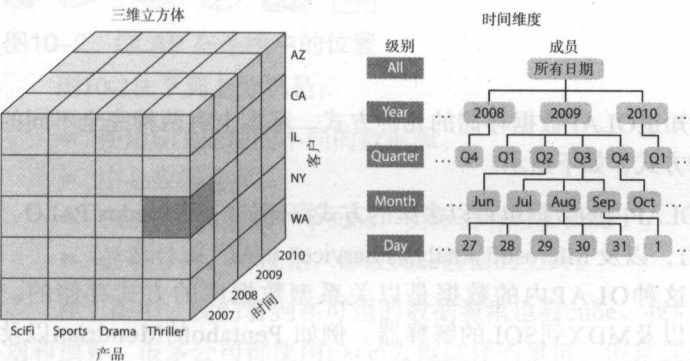


图10-1 时间维度和一个三维cube

图10-1显示了一个包含时间、产品和客户维度的OLAP cube。深色的单元是2007年、Thriller这种产品、伊利诺伊州的所有客户，这三个维度数据的交叉点。如果交叉点单元格里的数据是销售收入数据，我们就可以得到2007年，Thriller这个产品在伊利诺伊州的销售收入情况。这个销售数据是一个聚集数据，因为我们从图10-1右侧的时间维度可以看出：一年包含了几个季度，季度包含了月，月包含了天，这就是层次（hierarchy），一个维度可以包含多种层次，也就是数据展开的路径。同样客户维度也有层次关系（州→城市→地区→客户），产品维度也有层次关系（类别→产品组→产品）。

使用 OLAP cube来做分析的价值不在于这种多维的结构，而是在于预先聚集（pre-aggregation）这一个概念。OLAP cube里的销售收入数据不只是存储为最细粒度的数据（如某一个具体的客户和某一天），它还可以预先计算，在更高层次上做聚集，这样适合于交互、数据的即席查询。OLAP 工具的成功得益于下面几个因素。

- 快速的分析查询性能：因为数据按照维度进行了预先计算和聚集，所以能提供非常快的响应速度。
- 上卷和下钻的能力：因为有了这种层次结构，所以OLAP工具都可以提供上卷到上一层

和下钻到下一层的功能。

- **切片和组合的能力**: 可以选择不同的维度组合, 选择某个特定维度的值进行切片查看, 或者行列交换。
- **多维模型易于理解**: 大多数人都能理解这种通过维度和层次来分析数据的方法, 这也是为什么关系型数据仓库的多维模型可以变得这么流行的原因。

因为上面的这些特点, 再加上Microsoft Analysis Services颠覆性的价格体系, 使OLAP工具和技术越来越多地被采用。有一些OLAP产品还允许用户回写数据或创建不同版本的数据片(如预算A、预算B), 这样可以使数据库完成计划、预算、预测等工作。

到目前为止看上去都很好, 但是如何从 OLAP cube 里集成数据呢? 这也是OLAP系统面临的挑战。和分析工具不同, ETL工具并不能处理多维、层次结构的数据。所以我们需要把OLAP数据转换成行列结构的数据, 以便ETL工具可以处理。我们需要一种可以“扁平化”cube的方法。OLAP面临的另外一个挑战是如何获得不同聚集层的数据。尽管MDX可以让你做很多事情, 但把数据从cube里取出只是事情的开始。在大多数情况下, 还要做行列互换等操作, 以便适应其他数据源的要求。最后, 每个OLAP服务提供商都按照自己定义的格式提供数据, 所以解决了Mondrian的问题并不意味着解决了Microsoft Analysis Services的问题。

10.1.1 OLAP 存储类型

为了处理OLAP数据, 我们需要知道OLAP数据存储的几种方式。基本上有两种完全不同的方式, 另外还有介于两者之间的一种方式, 如下所示。

- **MOLAP**: 多维OLAP, 这种OLAP内的数据也是以多维的方式存储的。例如Jedox PALO、IBM-Cognos Powerplay或TM1, 以及 Microsoft Analysis Services (从广义上说)。
- **ROLAP**: 关系型OLAP, 这种OLAP内的数据是以关系型数据库的方式存储的。OLAP 引擎只是一个缓存, 以及MDX到SQL的解释器。例如 Pentaho的Mondrian以及 Microstrategy (从广义上说)。
- **HOLAP**: 混合OLAP, 细粒度的数据存储在关系型数据库中, 聚集后的数据存储于OLAP数据库中。上面提到的一些产品其实就是这种混合OLAP。Mondrian可以在内存中缓存聚集后的多维数据, 这就是混合OLAP的工作方式。其他产品如Microsoft的Analysis Services可以让你自己决定有多少数据预先聚集放在cube里、多少数据放在底层的数据库里。

这些存储方式的不同会影响到ETL的设计。对于MOLAP这种存储方式来说, 我们不得不使用MDX查询来获取数据, 除非有其他获取数据的方式。对ROLAP存储方式则不同, ROLAP数据是存在关系型数据库中的, 可以直接使用SQL, 而不是MDX来抽取数据。但这种直接通过SQL的方式会存在一些问题; ROLAP是用于大数据量查询的一种技术, 如果绕过了ROLAP引擎, 直接使用SQL, 可能会带来严重的性能问题。另外, 作为ROLAP的数据库里一般都会有事先计算好的聚集表, 来提供OLAP的性能, 如果ETL开发人员没有注意到这些聚集表, ETL过程就会给数据库带来沉重的负担。所以使用ETL从OLAP系统抽取数据的结论就是: 无论是ROLAP、MOLAP还是HOLAP, 最安全和最快速的方式就是使用MDX去查询, 尽量不要从关系型数据库中直接获取数据。

写数据则是另一回事, 不是所有的OLAP服务都能支持回写的。对Mondrian来说, 回写就意味着如果有了新数据, 这些新数据必须要被插入或更新到底层数据库中。然后还要重建缓存,

这样客户端才能看见更新的数据。

10.1.2 OLAP在系统中的位置

在大多数情况下，OLAP数据库被用作数据集市，也有的企业把OLAP作为数据仓库，但这是少数，不是通常情况。一个典型的使用OLAP技术的架构如图10-2所示。

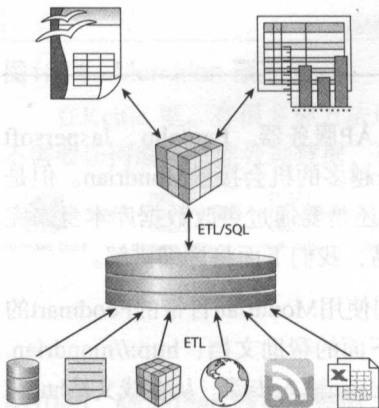


图10-2 OLAP 在系统中的位置

图10-2从下到上分别是：

- 使用ETL处理不同的数据源。
- 中心数据仓库。
- OLAP（抽象成一个cube，实际会有很多cube）。
- 前端分析，可视化，修改cube数据的工具。

在上图中，我们看到在可用的数据源里也有cube。我们前面也讲过，OLAP技术可以用于计划和预算。很多公司都使用Excel去做这样的事情，但是这样缺少了中心管理和安全权限，而像Palo这样的OLAP工具可以提供这些功能。所以此时，OLAP cube可以有两个功能：作为中心数据仓库来储存和管理信息，还可以作为一个数据源。图10-2中描述了OLAP的这两个作用，以及和这两种OLAP cube交互的工具。像Palo这样的产品还提供了Excel和OpenOffice Calc的插件，这样用户可以直接使用这些工具进行多维分析。

10.1.3 Kettle OLAP选项

除了能访问各种数据库，Kettle也可以从各种OLAP数据源中抽取数据，甚至可以回写到部分OLAP数据源中。下面是三个处理OLAP数据源的步骤。

- **Mondrian 输入步骤：**使用MDX 语句从Mondrian输入数据。因为这个步骤使用了Mondrian Schema里定义的数据库连接，所以不用再单独建立连接。另外要设置好Mondrian的Schema，Mondrian的Schema 定义了cube的结构。
- **OLAP 输入步骤：**这是一个通用的OLAP 输入步骤，可以从任意兼容XML/A协议的OLAP服务中抽取数据。目前几乎所有的OLAP服务都支持这一协议。在这个步骤中使用了两种技术：XML/A和Olap4J。后者是一个通用的开源库，用于在Java环境下使用MDX语句连接OLAP服务器。XML/A是客户端和OLAP服务通信的协议，它基于HTTP的

SOAP服务，它不仅读取数据，还可以用来写入数据和激活一些任务。

- **Palo 插件：**这个插件可能是功能最丰富且最易于使用的OLAP插件。这个插件包含了四个步骤来读写数据，更新或刷新维度信息，创建cube 里的新维度。

下面的章节分别以Mondrian、Microsoft SQL Server、Palo为例，来介绍上面的三个步骤。我们需要事先安装这三种OLAP服务器。

10.2 Mondrian

在任何一个开源BI的解决方案里，Mondrian都是默认的OLAP服务器。Pentaho、Jaspersoft和SpagoBI 都使用Mondrian 作为OLAP服务器。所以，你有越来越多的机会接触Mondrian。但是到目前为止，Mondrian 还没有回写功能，所以想要更改cube，还是要通过更改数据库本身来完成，然后再刷新Mondrian的缓存。从Mondrian 获取数据非常容易，我们下面将详细讲解。

在运行下面的例子之前，需要先安装并运行Mondrian。我们使用Mondrian自带的Foodmart的例子，如果你没有Mondrian和Foodmart样例数据库，可以参考下面的帮助文档：<http://mondrian.pentaho.org/documentation/installation.php>。转换的第一步就是Mondrian输入步骤。从在线文档<http://wiki.pentaho.com/display/EAI/Mondrian+Input>，可以看出这个输入步骤里有一个查询的例子。这个查询例子使用Foodmart里的Sales cube。我们看到在Mondrian输入步骤里有以下三个地方需要设置。

- **连接：**这是一个普通的关系型数据库连接，Mondrian 用来获取数据。
- **Mondrian Schema文件的位置：**指定Mondrian Schema文件的位置。
- **MDX查询：**定义用来查询的MDX 语句。

图10-3显示了Mondrian输入步骤的例子。

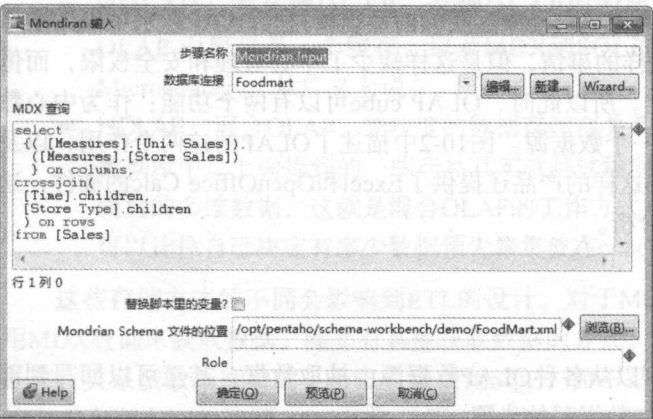


图10-3 Mondrian 输入步骤

使用“预览”功能就可以立刻看到MDX查询的结果，如图10-4所示。

可以看到，这些输出结果并不能直接用来作为报表输出。我们还要做下面的一些处理：

- 去除所有的方括号。
- 将年和季度等字段分开。
- 抽取商店类型字段。

Examine preview data

Rows of step: Mondrian Input (24 rows)

| # | [Time] | [Store Type] | [Measures].[Unit Sales] | [Measures].[Store Sales] |
|---|--------------------|--|-------------------------|--------------------------|
| 1 | [Time].[1997].[Q1] | [Store Type].[All Store Types].[Deluxe Supermarket] | 20977.0 | 43864.84 |
| 2 | [Time].[1997].[Q1] | [Store Type].[All Store Types].[Gourmet Supermarket] | 3822.0 | 8203.89 |
| 3 | [Time].[1997].[Q1] | [Store Type].[All Store Types].[HeadQuarters] | | |
| 4 | [Time].[1997].[Q1] | [Store Type].[All Store Types].[Mid-Size Grocery] | 3096.0 | 6439.32 |
| 5 | [Time].[1997].[Q1] | [Store Type].[All Store Types].[Small Grocery] | 1457.0 | 3037.78 |
| 6 | [Time].[1997].[Q1] | [Store Type].[All Store Types].[Supermarket] | 36939.0 | 78082.52 |
| 7 | [Time].[1997].[Q2] | [Store Type].[All Store Types].[Deluxe Supermarket] | 16371.0 | 34486.06 |
| 8 | [Time].[1997].[Q2] | [Store Type].[All Store Types].[Gourmet Supermarket] | 5837.0 | 12597.15 |

Close

图10-4 Mondrian 预览输出

在Kettle 里，有很多种方法可以做上面的转换，其中一种方法就是图10-5所示的转换流程，不需要任何编码就能处理数据，得到想要的结果。

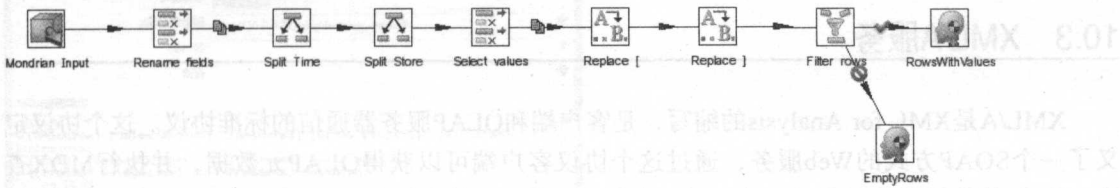


图10-5 Mondrian 转换流程图

处理Mondrian数据的第一步就是给字段重命名，便于以后的流程使用这些字段。接下来是两个“拆分字段”步骤把一个字段里的数据根据指定的分隔符拆分成多个字段，并指定新的字段名。图10-6是拆分Time字段的截图。

拆分字段

步骤名称: Split Time

需要拆分的字段: time

分隔符: .

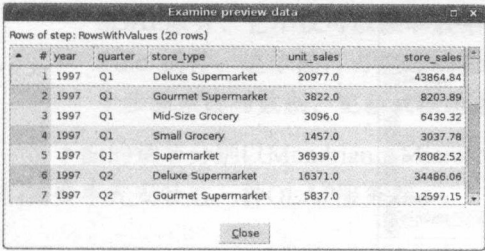
| # | 新的字段 | 标识符 | 移除标识符 | 类型 | 长度 | 精度 | 格式 | 分组符号 | 小数点符号 | 货币符号 | 满足条件时置空 | 缺省 | 去除空格类型 |
|---|---------|-----|-------|--------|----|----|----|------|-------|------|---------|----|--------|
| 1 | time | 是 | | String | | | | | | | | | 不去掉空格 |
| 2 | year | 否 | | String | | | | | | | | | 不去掉空格 |
| 3 | quarter | 否 | | String | | | | | | | | | 不去掉空格 |

确定(O) 取消(C)

图10-6 拆分Time字段的步骤

另一个拆分字段步骤拆分的是Store字段。当然也可以使用其他步骤达到同样的效果。例如“字符串剪切”步骤，可以获得字符串里的一个子串。但这个方法可能会有些问题，例如“All Store Types”被改成了“All Types”，那么使用“字符串剪切”步骤就会出现错误。而“拆分字段”步骤则更健壮。后面的“字段选择”步骤去掉一些没有用的字段，再后面的两个“字符串替换”步骤移去了数据里的方括号。为了完整，我们最后还加了一个“过滤”步骤，过滤没有收入数据的Store 类型，如总部。在本书第10章的例子里可以找到这个转换www.wiley.com/go/kettlesolutions。

当然，可以使用Java脚本步骤实现上面所有这些转换。但是写代码相对麻烦一些，例如为了获取年份，你可能要写这样的代码 `var year= substr(time,indexOf(time,"")+2,4)`；要获得季度，还要用嵌套的indexOf方法。所以你完全可以避免写这些烦琐的代码。这个转换运行的结果是我们想要的结果，如图10-7所示。



| # | year | quarter | store_type | unit_sales | store_sales |
|---|------|---------|---------------------|------------|-------------|
| 1 | 1997 | Q1 | Deluxe Supermarket | 20977.0 | 43864.84 |
| 2 | 1997 | Q1 | Gourmet Supermarket | 3822.0 | 8203.89 |
| 3 | 1997 | Q1 | Mid-Size Grocery | 3096.0 | 6439.32 |
| 4 | 1997 | Q1 | Small Grocery | 1457.0 | 3037.78 |
| 5 | 1997 | Q1 | Supermarket | 36939.0 | 78082.52 |
| 6 | 1997 | Q2 | Deluxe Supermarket | 16371.0 | 34486.06 |
| 7 | 1997 | Q2 | Gourmet Supermarket | 5837.0 | 12597.15 |

图10-7 Mondrian 结果输出

对于Mondrian来说，还有另外一个方法。Mondrian 也支持XML/A协议，也就是说可以直接使用“OLAP输入”步骤从Mondrian 获取数据。这是我们下一节的内容。

10.3 XML/A服务

XML/A是XML for Analysis的缩写，是客户端和OLAP服务器通信的标准协议。这个协议定义了一个SOAP方式的Web服务，通过这个协议客户端可以获得OLAP元数据，并执行MDX查询。XML是数据交换的格式。在SOAP（简单对象访问协议，Simple Object Access Protocol）里封装了MDX查询（实际这种协议可以支持MDX和SQL查询）。关于XML/A协议参考：<http://www.xmla.org>。

为了支持XML/A 协议，Kettle 提供了“OLAP 输入”步骤，这个步骤使用开源的Olap4J包。“OLAP 输入”步骤考虑到了XML/A和Olap4J的各种复杂情况，所以使用这个步骤只要考虑如何输入正确的连接串和编写MDX查询就可以了。因为有这个标准，可以访问的OLAP数据源一下就会多起来，实际各种OLAP服务，无论是开源的还是商业的，都可以使用XML/A来访问。例如SAP的BW、Oracle的Hyperion Essbase、IBM的Cognos TM/1、Mondrian和Microsoft Analysis Services。

为了使用XML/A，需要将OLAP服务设置为可以接收HTTP 请求。我们的例子使用了Microsoft SQL Server 2008 Analysis Services（MSAS），另外服务器还要做一些配置才能接收XML/A请求。这些配置不在本书介绍的范围内，我们只提供下面的一些链接供读者参考。

- 要在Windows Server 2003 上配置SQL Server 2005 Analysis Services，参考<http://technet.microsoft.com/en-us/library/cc917711.aspx>。
- 要在Windows Server 2008 上配置 SQL Server 2008 Analysis Services，参考 <http://bloggingabout.net/blogs/mglaser/archive/2008/08/15/configuring-http-access-to-sql-server-2008-analysis-services-on-microsoft-windows-server-2008.aspx>。
- 要在Windows Server 2003 上配置 SQL Server 2008 Analysis Services，参考第一个链接。
- 可以通过 HTTP 测试是否可以连上MSAS服务。如使用下面的URL，<http://<server address>/olap/msmdpump.dll>，<server address> 要替换成实际的机器名或IP地址。如果能得到类似下面的响应：XML/AnalysisError.0xc10e0002Parser: The syntax for ‘GET’ is incorrect，就说明HTTP 访问正常。
- 在配置上最难实现的就是给cube数据分配正确的用户权限，关于分配用户权限参考下面的链接：http://www.activeinterface.com/b2008_12_29.html。
- 如果你只是想检查XML/A是否工作正常，而不用关心安全问题，可以在IIS里将OLAP

Web应用设置为可以匿名访问。然后，可以在MSAS数据库里添加一个角色，并为这个角色赋予一定权限，然后给这个角色添加一个用户。可以把匿名登录的用户分配给这个角色。通常这种用户的账户是IUSER_<machine name>，把machine name替换为你服务器的名字。

现在你就可以试着从Kettle访问OLAP服务器了。新建一个转换，在“OLAP 输入”步骤里输入XML/A的URL；如果你设置了匿名访问，用户名和密码可以都不填。要连上数据库，还需要输入Catalog的名字。这就是SQL Server企业管理器里MSAS数据库的名字，包括命名空间。然后就可以设置OLAP cube里需要哪个数据。我们使用一个相对简单的数据集，年和产品类别作为行，销售金额和毛利润作为列。“OLAP输入”步骤的完整配置如图10-8所示。

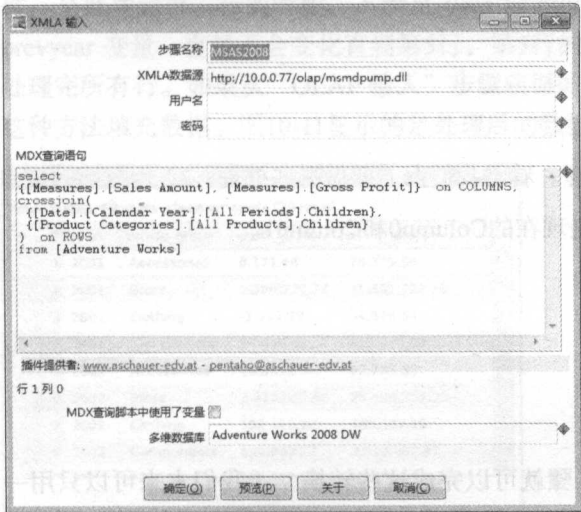


图10-8 OLAP 输入步骤

我们此时也可预览一下结果，预览结果的窗口如图10-9所示。

警告：在编写本书时，“OLAP输入”步骤会自动触发一个process命令，强制要求cube在返回结果前先重构，所以每次执行都有几分钟的时间。

| # | Column0 | Column1 | [Sales Amount] | [Gross Profit] |
|----|---------|-------------|-----------------|----------------|
| 1 | CY 2001 | Accessories | \$20,235.36 | \$8,171.48 |
| 2 | | Bikes | \$10,661,722.28 | \$1,580,237.72 |
| 3 | | Clothing | \$34,376.34 | (\$1,911.73) |
| 4 | | Components | \$615,474.98 | \$54,035.47 |
| 5 | CY 2002 | Accessories | \$92,735.35 | \$28,351.25 |
| 6 | | Bikes | \$26,486,358.20 | \$2,413,803.00 |
| 7 | | Clothing | \$485,587.15 | \$102,113.92 |
| 8 | | Components | \$3,610,092.47 | \$425,983.77 |
| 9 | CY 2003 | Accessories | \$590,242.59 | \$283,673.27 |
| 10 | | Bikes | \$34,910,877.69 | \$3,050,962.76 |
| 11 | | Clothing | \$1,010,112.16 | \$154,607.64 |
| 12 | | Components | \$5,482,497.29 | \$414,831.73 |
| 13 | CY 2004 | Accessories | \$568,844.58 | \$314,271.16 |
| 14 | | Bikes | \$22,561,568.03 | \$3,470,093.14 |
| 15 | | Clothing | \$587,537.80 | \$114,026.17 |
| 16 | | Components | \$2,091,011.92 | \$138,015.51 |
| 17 | CY 2006 | Accessories | | |
| 18 | | Bikes | | |

图10-9 OLAP 输入步骤的预览结果

如果直接在Microsoft Analysis Services上执行 MDX查询，我们会发现时间列没有排重，如图10-10所示。

| Messages | | Results | |
|----------|-------------|-----------------|----------------|
| | | Sales Amount | Gross Profit |
| CY 2001 | Accessories | \$20,235.36 | \$8,171.48 |
| CY 2001 | Bikes | \$10,661,722.28 | \$1,580,237.72 |
| CY 2001 | Clothing | \$34,376.34 | (\$1,911.73) |
| CY 2001 | Components | \$615,474.98 | \$54,035.47 |
| CY 2002 | Accessories | \$92,735.35 | \$28,351.25 |
| CY 2002 | Bikes | \$26,486,358.20 | \$2,413,803.00 |
| CY 2002 | Clothing | \$485,587.15 | \$102,113.92 |
| CY 2002 | Components | \$3,610,092.47 | \$425,983.77 |
| CY 2003 | Accessories | \$590,242.59 | \$283,673.27 |
| CY 2003 | Bikes | \$34,910,877.69 | \$3,050,962.76 |
| CY 2003 | Clothing | \$1,010,112.16 | \$154,607.64 |
| CY 2003 | Components | \$5,482,497.29 | \$414,931.73 |
| CY 2004 | Accessories | \$568,844.58 | \$314,271.16 |
| CY 2004 | Bikes | \$22,561,568.03 | \$3,470,093.14 |
| CY 2004 | Clothing | \$1,593,593.00 | \$314,026.17 |

图10-10 MSAS 上的查询结果

可以使用 Kettle对图10-9 里的数据再做处理。看看我们都需要做哪些处理：

- 字段重命名，让字段名更有意义而不是现在的Column0和Column1。
- 删除字段名里的方括号。
- 删除 \$ 符号。
- 左圆括号替换为减号。
- 删除右圆括号。
- 填充上空的年份数据。

这看上去很复杂，但实际上只要用两个步骤就可以完成这些转换。（我们本来可以只用一个，但是用两个可以使转换更容易理解和维护）。

第一个步骤非常简单：只是一个“Rename fields”步骤，用来给所有的字段重命名。列表里剩下的工作需要放在一个 Java脚本步骤里完成（见本章的MSAS_Example.ktr 转换）。

```
var prevyear;

if (gross_profit != null) ~
var gross_profit = gross_profit.replace("$", "-").
replace("$", "").replace(")", "");

if (sales_amount != null) ~
var sales_amount = sales_amount.replace("$", "-").
replace("$", "").replace(")", "");

if (year != null) var year = year.replace("CY ", "");

if (year != null)
{
    prevyear = year;
}
else
{

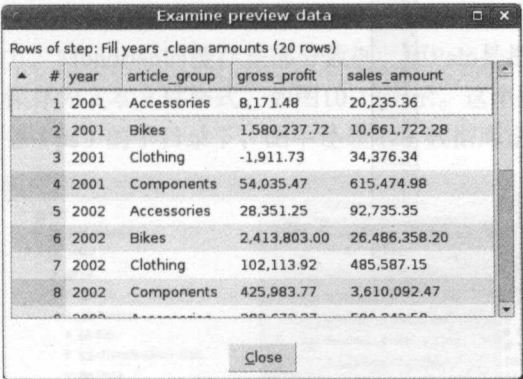
```



```
year = prevyear;
}
```

代码先定义了一个变量 `prevyear`，后面用这个变量存储前一数据行的年份。后面的三行使用 `string.replace(search_string,replace_string)` 方法来替换字段里的部分值。我们看到可以对一个字段多次调用 `replace` 方法，这样处理字符串非常灵活。最后一部分需要详细解释一下。

我们要知道JavaScript是一行一行处理数据的。这就是说可以设置一个变量，只要设置变量的条件不发生变化，变量的值也不会发生变化。但要注意的是：在这个例子里，我们已经知道输入数据是什么样子的，我们也知道数据的第一行包括了一个年份数据。根据后面的数据和排序规则，可以把第一行的年份赋值给 `prevyear` 变量，因为第一行数据满足了 `year != null` 这个条件。当处理到第二行的时候，不满足 `year != null` 这个条件，所以执行了 `else` 的分支。接下来，`prevyear` 变量一直都不会变化直到第5行，第5行的年份字段又赋给了 `prevyear` 变量。就这样直到处理完所有行。如果从“OLAP 输入”步骤获得的数据里很多字段里都有空值，我们就可以使用这种方法填充数据。图10-11显示的是处理后的数据，可以给数据仓库使用了。



| # | year | article_group | gross_profit | sales_amount |
|---|------|---------------|--------------|---------------|
| 1 | 2001 | Accessories | 8,171.48 | 20,235.36 |
| 2 | 2001 | Bikes | 1,580,237.72 | 10,661,722.28 |
| 3 | 2001 | Clothing | -1,911.73 | 34,376.34 |
| 4 | 2001 | Components | 54,035.47 | 615,474.98 |
| 5 | 2002 | Accessories | 28,351.25 | 92,735.35 |
| 6 | 2002 | Bikes | 2,413,803.00 | 26,486,358.20 |
| 7 | 2002 | Clothing | 102,113.92 | 485,587.15 |
| 8 | 2002 | Components | 425,983.77 | 3,610,092.47 |

图10-11 处理后的MSAS数据

对这个例子来说，使用“OLAP 输入”步骤是最灵活的方法，也是标准的方法。几乎每个OLAP服务都可以使用这种方法访问，可以使用一个标准的方法访问这些服务器获取和转换数据。标准化为我们构建ETL解决方案提供了便利。

10.4 Palo

Palo是一个开源的内存多维数据库，由德国的软件公司Jedox (<http://www.jedox.com>) 开发。（如果想知道这个名字的起源，你把名字倒过来就知道了。）我们假设你的Palo服务器正在运行；关于其他机器如何能访问Palo服务器，请阅读下面的提示。关于Palo的安装和配置，请访问Palo的网站，下载开源版本的Palo。

提示：要让其他机器能访问Palo服务器，需要把http项加入到palo.ini文件中，palo.ini文件位于Palo的Data目录下。默认的设置是http "127.0.0.1" 7777，就是说服务器只允许本地的访问。可以改一下，或增加一项。如果你使用了多个项，要确保每一项使用不同的端口号。例如，我们使用的palo.ini文件有下面几项：

```
http "127.0.0.1" 7777
http "10.0.0.71" 7778
```

本章的例子使用的是2010年4月发布的Palo 3.1和它自带的Demo数据库。有几种方法可以浏览Palo cube里的数据。默认的Palo for Excel 包含了一个Excel 插件，可以用来查看Palo里的数据。如果你用的是Open Office，可以使用由 Tensegrity 公司（http://www.jpalo.com/en/products/palo_open_office.html）提供的PalOOCa 插件。图10-12显示的Palo Modeler也是Open Office的插件的一部分。

在图里，演示数据库已经打开，演示数据带着产品维度。产品维度的层次分为三级：所有产品、产品组和产品。后面的Kettle 需要这些信息，因为Kettle里没有预览Palo cube数据的功能。

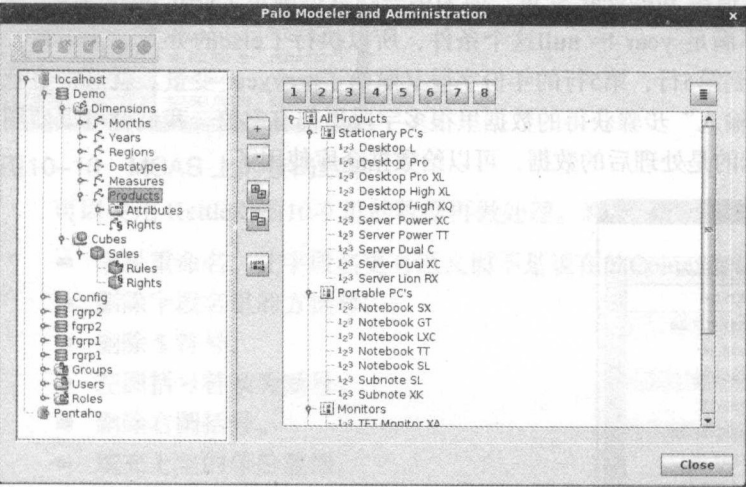


图10-12 Palo Modeler的演示数据

10.4.1 建立Palo 连接

首先要获取由jPalo（<http://www.jpalo.com>）发布的客户端连接软件。客户端叫Palo Java API，可以从下面的链接下载：<http://www.jpalo.com/en/products/startdownload.php?product=API&lang=en>。

另外在SourceForge上也能下载到 jPalo（<http://sourceforge.net/projects/jpalo/>），但SourceForge上的版本通常都旧一些。当然也可以从代码库下载 trunk 分支，然后自己编译。

说明：trunk是指代码库中最新的分支。关于trunk的更多信息参考 [http://en.wikipedia.org/wiki/Trunk_\(software\)](http://en.wikipedia.org/wiki/Trunk_(software))。

下载后解压缩 .zip 文件，然后把 lib 目录下的jpalo.jar文件复制到Kettle的libext 目录下。

此时Palo步骤就可以连接到Palo cube上了。下面的几个步骤都可以用了，两个输入步骤，两个输出步骤。

- Palo 度量输入
- Palo 维度输入
- Palo 度量输出
- Palo 维度输出

在继续之前，我们要检查Palo服务器是否可以连通。新建或打开一个转换，右键单击数据库

连接，新建一个数据库连接，如果你是在本机上安装的Palo，填入下面的数据库连接参数：

| | |
|----------------|-----------|
| Host name: | localhost |
| Database Name: | Demo |
| Port Number: | 7777 |
| User Name: | admin |
| Password: | admin |

单击“测试”，如果正常，Kettle会告诉你连接成功。如果不正常，检查服务、参数或防火墙设置。

10.4.2 Palo 架构

在读写Palo cube之前，我们要了解一些Palo内部工作的原理。尽管Palo是一个内存OLAP数据库，但并不意味着它没有持久层。和其他数据库一样，数据都是存储在硬盘上的，Palo和Mondrian有些类似，它们都要在启动时把数据从硬盘加载到内存里，这也是它们唯一相似的地方。Mondrian的缓存是部分数据，而Palo是把数据库的内容都加载到内存里。Palo在硬盘的数据保存为文本文件格式，如图10-13所示。这个图显示了一个默认的Linux安装，数据保存在Palo服务（ps）的子目录下，图中还显示了数据库定义文件的前面部分。

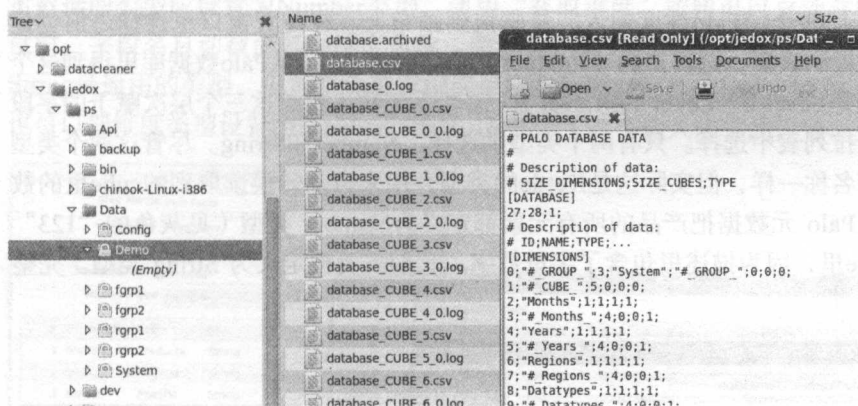


图10-13 Palo数据文件

因为数据保存成文本文件，所以做数据库备份或拷贝比较容易。停止Palo服务器，压缩并保存数据目录再移动到其他位置就可以做备份了。如果想把一个数据库移动到另一个Palo服务器下，停止目的机器的Palo服务，把完整的目录放到Data目录下，并重新启动Palo服务。Palo的建模工具就能看到复制过来的数据库了。

Palo还有几个特点可以帮助我们理解Kettle如何从Palo读写数据。首先，Palo里没有预先定义好的维度或维度类型。大部分OLAP服务都把普通维度和时间维度区分开，也把各个度量区分开。而Palo没有区分普通维度和日期维度：每个维度的选项都是一样的。Palo里的每个元素只有两种数据类型：String和Number（元素是Palo里cube最基本的单元，是最细粒度的数据）。在使用Kettle从Palo cube里抽取数据时，你要指定从Palo读写的数据是String还是Number类型，在指定类型时有几个要注意的地方：例如，Month元素，在Palo里是Number类型，但是如果在Kettle的“Palo 维度输入”步骤里，把它指定为Kettle的Number类型，就会报类型转换错误。因为没有日期维度，所以可以通过其他方式来构建日期维度。我们观察日期维度，可以发现日期维度包含

了年和月维度，但它们是独立的维度。如果使用独立的年月维度，我们也可以构建交叉报表，年作为X轴，月作为Y轴，而使用独立的日期维度，做不到这样的效果。

Palo数据库也包括了属性。属性用来存储元素的一些额外信息。例如，演示数据库里，Product 维度里包含Price per Unit属性。这样对每个产品，都可以存储价格信息。目前，Kettle还不能抽取这些属性信息。如果想要这些信息，可以利用Palo插件，把这些数据粘贴到电子表格里，然后使用Kettle 读电子表格里的数据。

最后还要注意的一个Palo的特点是细粒度的授权。每个用户属于一个用户组，每个用户组有一到多个角色。角色决定用户能做什么事情（创建其他用户、删除cube、定义规则，等等）；用户组能决定用户对数据的访问权限。如果能用admin账号来访问，可能没什么权限问题。但在生产环境中，ETL很少能用admin账号去访问数据，所以要注意所用的账号的权限。

10.4.3 读Palo数据

在Kettle 里有两个Palo数据的读取步骤，“Palo 度量输入”和“Palo 维度输入”前面的输入步骤可以获取最细粒度的Palo cube里的事实数据，而后面的输入步骤只能获取到维度数据，但一次只能获取一个维度。如果要抽取全部cube和全部的维度及层次，要给每个维度都创建一个维度输入步骤，以及一个度量输入步骤。让我们先看看如何使用“Palo 维度输入”步骤抽取产品维度。

创建一个新转换，把一个“Palo维度输入”步骤拖到画布上。在配置窗口里选择之前创建的连接，然后在维度下拉列表里，选择产品（Product）维度。Kettle 会从Palo数据库里得到这个维度的三个层次，分别是Level 0、Level 1和Level 2。需要手工输入想给这三个层次赋予的字段名，字段类型可以从下拉列表中选择。只有两个类型供选择，Number和String。尽管这两个类型看上去和cube里的类型名称一样，但实际这是Kettle的类型，用来自动转换读取到的cube里的数据。在我们的例子里，Palo 元数据把产品的所有元素都定义为Number 类型（见灰色的“123”指示图标），而在Kettle里，因为描述里包含了字母数字，它们需要被定义为 String 类型。完整的配置如图10-14所示。

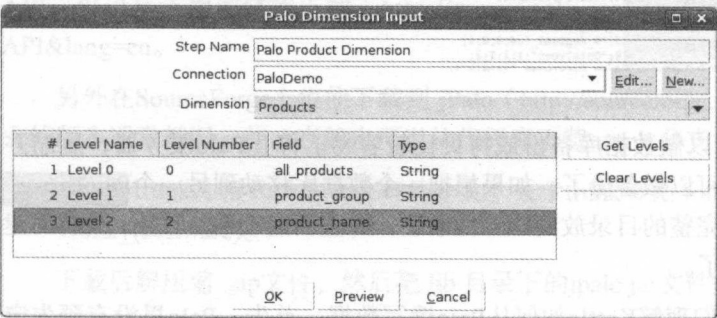


图10-14 Palo 维度输入配置

图10-15 显示了数据预览窗口，实际上就是一个三层的维度表，没有主键。为了使“维度查询”步骤能识别出维度的变化，我们要把product_name列作为维度主键，所以维度表里只有all_products和product_group 列里的数据可以变化。为了把“Palo 维度输入”步骤取出的维度数据转换成关系型的产品维度表，我们需要使用“增加序列”步骤增加一个代理键，然后使用“增加常量”步骤再分别增加“date_from”、“date_to”和“is_current”字段。

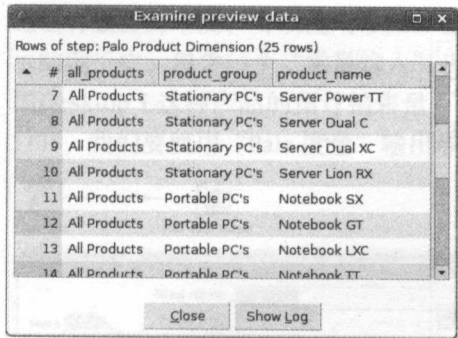


图10-15 Palo 维度输入步骤预览

下面的工作要从Palo数据库里抽取事实数据。把“Palo Cells输入”步骤拖曳到画布上，设置要从哪个cube里抽取数据。如果你使用的是admin账号，cube 下拉列表里也会显示出所有系统cube，但我们这里只关心第一个。我们前面说过，cube是各个维度层次的交点。在Palo里cube中的数据是最低层细粒度的数据，它们在运行时才做聚集。所以我们用Kettle抽取到的是最细粒度的数据。

在Demo数据库的Sales cube里，度量数据有六个维度，我们要给cube的度量数据起一个名字，因为在Palo里度量数据没有自己的名字；度量数据的列名在“度量名”输入框里设置。度量数据的类型应设置为Number类型，使用“获取维度”按钮可以自动获取到维度信息。获取维度时，字段名自动被命名为维度名称，需要的话可以修改字段名称。另外还要设置维度类型，String是通用的类型，但如果你确定维度里所有的类型都是数值类型（如Sales里的Year字段），也可以把维度类型设置为Number。图10-16 显示了这个输入步骤的设置以及预览窗口里的数据。

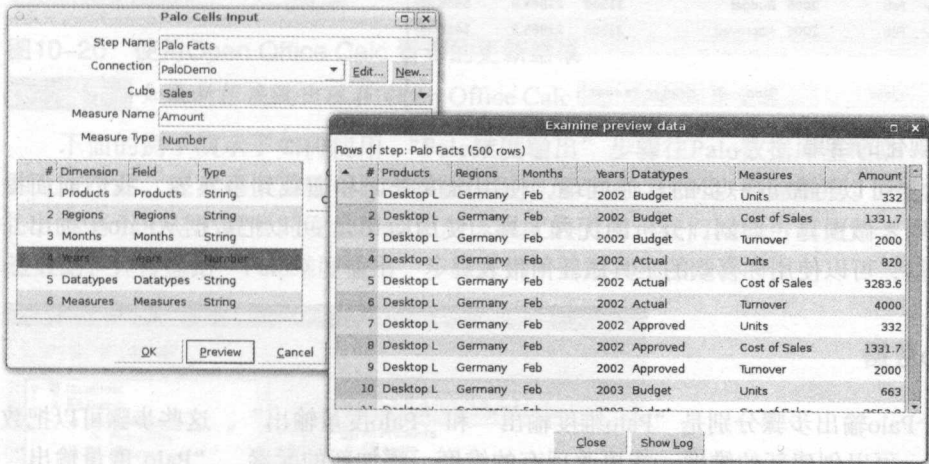


图10-16 “Palo 度量输入” 配置和预览

从上图可以看到，预览的是最细粒度的，最低层次的数据。我们看到图10-16里的数据还不能直接被用作报表数据。因为销售数量、销售金额、销售成本等数据分散在不同的数据行里，所以要先做一下旋转，使用行转列（反正规化）步骤。“行转列”步骤需要设置一个“关键值字段”（旋转字段），在这个例子就是Measures字段。“分组字段”就是除了Amount字段和Measures字段之外的所有字段，Amount字段将被作为“值字段”。“目标字段”要手工设置，可以设置为任意的名字。最后的配置窗口如图10-17所示。

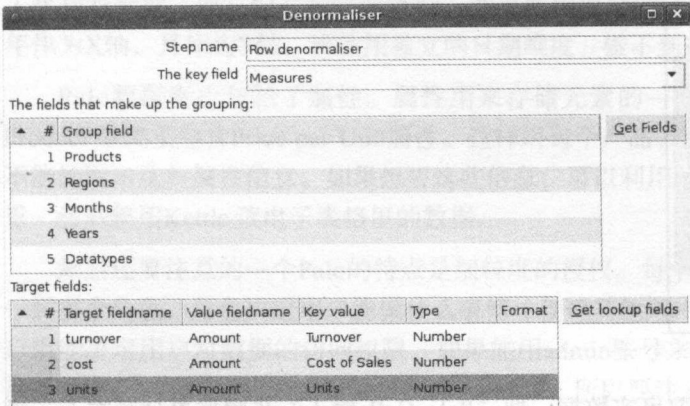


图10-17 Palo数据的反正规化

因为所有的数据已经是最细粒度的数据，这里不用再做聚集处理。要测试运行结果，可以预览一下，预览结果如图10-18所示。

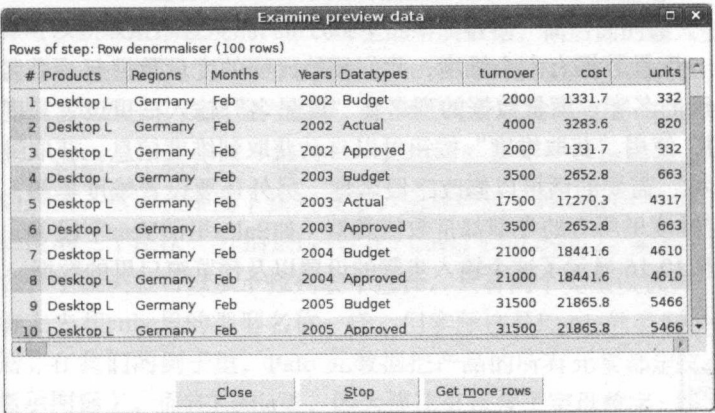


图10-18 反正规化的结果

现在这些数据可以再做进一步的其他处理，使用维度查询和加载到事实表。我们前面提过，Palo是一个用来做预算、计划、分析的优秀工具。使用Kettle，可以把数据从Palo中抽出并加载到数据仓库里，可以使用你喜欢的工具做任何报表。

10.4.4 写Palo数据

Kettle的两个Palo输出步骤分别是“Palo维度输出”和“Palo度量输出”。这些步骤可以把数据插入到cube里，可以创建新的维度，或更改现有的维度，添加新的元素。“Palo 度量输出”步骤要求准备输出的数据的结构要和已存cube的数据结构一致。我们准备把一个新的产品添加到产品维度里，然后再把这个新产品的度量数据加载到Palo cube里。我们不能一上来就加载度量数据，除非度量数据的维度和已有维度相同。所以加载新数据一般要分两步：首先加载维度数据，然后加载度量数据。

警告：在更新或加载新数据之前，要备份Palo数据库；如果把度量数据写入到不存在的维度里，Palo可能会崩溃。

下面我们通过例子说明Palo输出步骤。首先我们更新Palo数据库里已有的数据。我们要知

道已有数据的维度，可以使用带Palo 插件的Calc或Excel来查看已有数据的维度。在如图10-19所示的例子中，包含一个“常量输入”步骤，里面只有一行数据，这行数据用来更新2002年1月德国Desktop L这种产品的预计销售量。如果看过Demo数据库，可以知道这个维度的数据应该是1135，我们要使用“Palo 度量”输出步骤把这个数据改成99。

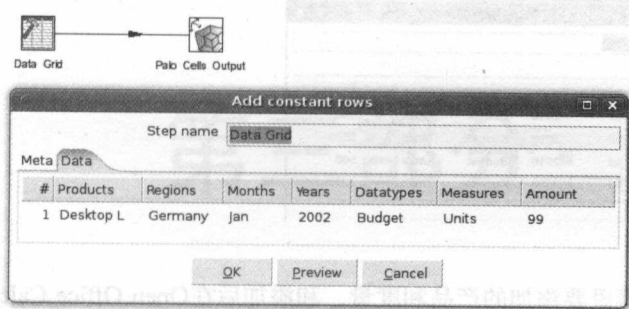


图10-19 更新Palo数据库

从图10-20的输出截图可以看到，这个数据已经被成功更新了。

| | A | B |
|----|-----------------|----------------|
| 1 | Database | localhost/Demo |
| 2 | Cube | Sales |
| 3 | Months | Jan |
| 4 | Years | 2002 |
| 5 | Datatypes | Budget |
| 6 | Measures | Units |
| 7 | | |
| 8 | Edit | Germany |
| 9 | Stationary PC's | 4,518 |
| 10 | Desktop L | 99 |
| 11 | Desktop Pro | 345 |
| 12 | Desktop Pro XL | 249 |
| 13 | Desktop High XL | 193 |

图10-20 使用Open Office Calc 看到的更新结果

提示：如果新维度没出现在 Open Office Calc 中，需要断开重连。

下面的例子演示了如何使用“Palo维度输出”步骤往Palo数据库里增加一个新的产品类别和几个产品。注意这个步骤可以在数据库里增加维度，也可以删除维度。删除维度的时候要小心，所有引用这个维度的度量将丢失对该维度的引用。我们同样可以使用“常量输入”步骤构造数据，并传递给“Palo维度输出”步骤。例子如图10-21所示。

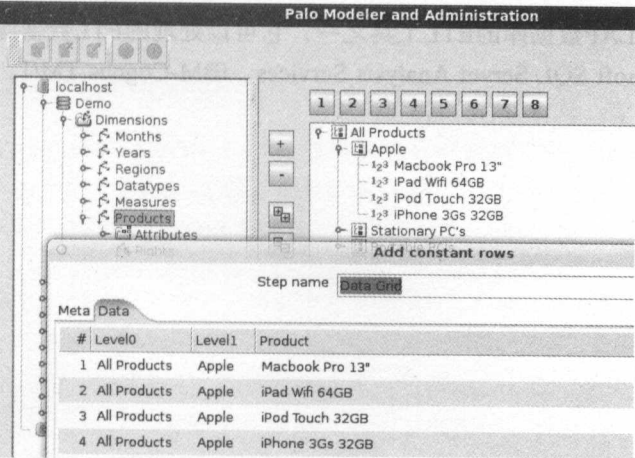


图10-21 添加一个新的产品类别和几个产品

在图10-21里，我们看到新的产品数据已经保存在Palo数据库中，增加完产品维度的数据后，我们还要给年维度增加一个值。增加完这两个维度数据后，我们就可以给新的产品增加度量数据了，如图10-22所示。

| A | | B | | C | | D | | E | | F | | G | | H | |
|----|-----------------|----------------|--|---------------------------|--|---|--|---|--|---|--|---|--|---|--|
| 1 | Database | localhost/Demo | | Add constant rows | | | | | | | | | | | |
| 2 | Cube | Sales | | Step name <u>Data.Gnd</u> | | | | | | | | | | | |
| 3 | Months | Jul | | Meta <u>Data</u> | | | | | | | | | | | |
| 4 | Years | 2010 | | | | | | | | | | | | | |
| 5 | Datatypes | Budget | | | | | | | | | | | | | |
| 6 | Measures | Units | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | |
| 8 | Est | Germany | | | | | | | | | | | | | |
| 9 | Apple | 5,400 | | | | | | | | | | | | | |
| 10 | Macbook Pro 13" | 150 | | | | | | | | | | | | | |
| 11 | iPad Wifi 64GB | 3,500 | | | | | | | | | | | | | |
| 12 | iPod Touch 32GB | 500 | | | | | | | | | | | | | |
| 13 | iPhone 3Gs 32GB | 1,250 | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | |

| # | Products | Regions | Months | Years | Datatypes | Measures | Amount |
|---|-----------------|---------|--------|-------|-----------|----------|--------|
| 1 | Macbook Pro 13" | Germany | Jul | 2010 | Budget | Units | 150 |
| 2 | iPad Wifi 64GB | Germany | Jul | 2010 | Budget | Units | 3500 |
| 3 | iPod Touch 32GB | Germany | Jul | 2010 | Budget | Units | 500 |
| 4 | iPhone 3Gs 32GB | Germany | Jul | 2010 | Budget | Units | 1250 |

图10-22 给新的产品添加预算数据

图10-22 显示了“固定常量输入”步骤里要添加的产品和度量，和添加后在Open Office Calc里看到的添加后的数据。从本节可以看到使用Kettle的几个 Palo步骤来处理Palo数据非常容易。

10.5 小结

本章主要围绕多维数据主题，也就是OLAP cube，展开介绍。本章的第一部分介绍使用OLAP的益处，OLAP为什么会作为商业智能或数据仓库解决方案的一部分。接下来我们提供了三个例子，演示了Kettle里的不同的OLAP步骤。我们介绍了下面的产品和技术。

- 使用“Mondrian输入”步骤，通过MDX从Mondrian cube中读取数据：把读取到的数据再通过“拆分字段”、“字段选择”、“替换字符串”和“过滤”步骤做进一步的处理。
- 使用“OLAP 输入步骤”从Microsoft SQL Server 2008 Analysis Services中读取数据：把读取到的数据再通过“Java Script”步骤做清洗处理，循环增加缺失的数据，以供后面的步骤使用。
- 使用四个Palo步骤从Palo中读取数据，或把数据写入到Palo数据库中：我们演示了如何反正规化数据，以及如何更新cube中的数据，如何增加新的维度，以及新维度下的度量数据。

Kettle是为数不多的可以处理各种OLAP数据库的ETL工具之一，它可以处理的ETL数据源包括Hyperion Essbase、SAP BW、Microsoft SQL Server Analysis Services、IBM-Cognos TM/1、Mondrian和Palo。

第11章 ETL开发生命期

在前面章节中,我们介绍了Kettle如何适用于Ralph Kimball提出的34种ETL子系统。本章将主题扩大到整个ETL开发生命期。同时也兼顾一些主题的细节。

ETL开发是构建和维护数据仓库的一个重要组成部分,所以ETL开发应该作为数据仓库建设过程的一部分,而不是一个独立的项目。一个项目往往有一个或多个预定义的目标和交付物,有明确的开始和结束点。一个过程往往是定期的重复的工作。创建ETL解决方案是项目的一部分,监控、维护、适配ETL解决方案都是过程的一部分。当然,适配已存在的ETL作业可以通过类似项目的方式完成。本章主要聚焦在生命周期的起点,也就是ETL方案的构建上。ETL方案包括分析、设计、构建、测试、文档和交付等阶段。当然我们要用尽可能低的成本快速解决这些挑战。Pentaho的敏捷BI工具可以很好地提供支持。

11.1 解决方案设计

任何解决方案都需要设计,直接跳到编码阶段是每个开发人员容易犯的错误。即使简单地在一个信封背后大概画一个草图,也会提高工作质量。而一种极端的情况是花了大量的时间去做设计,却没有开展实质性的工作。我们认为正确的方法应该在这两种情况之间。

11.1.1 好习惯和坏习惯

ETL设计工作应该在数据仓库设计之后才开始。有时开发人员习惯根据源系统的结构去调整数据仓库系统的结构。Kettle也支持这个工作方式,如图11-1所示,单击SQL按钮,可以生成修改表结构的SQL语句。

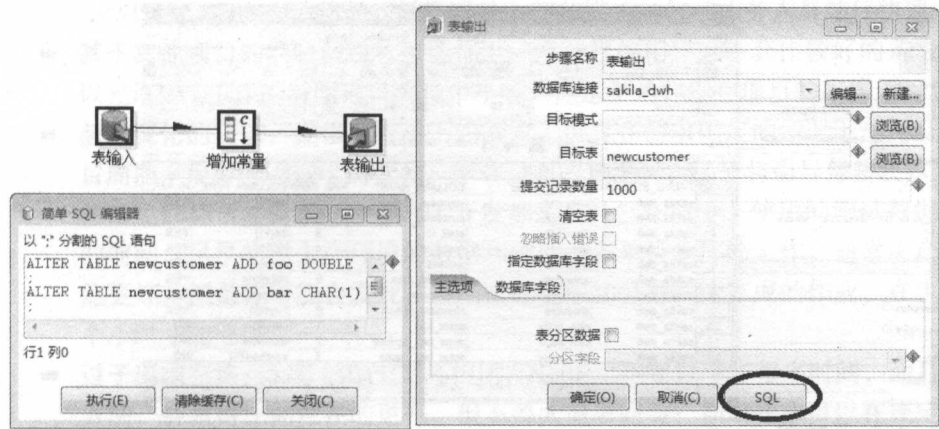


图11-1 表输出的修改表的语句

尽管这种方式也可以快速开发原型，但我们建议在生产环境中不要使用这种方式。一个好的数据仓库应该经过充分的设计，应该来源于业务需求而不是来源于源系统的数据结构。对数据仓库设计有兴趣的读者可参考Claudia Imhoff et al.的*Mastering Data Warehouse*（Wiley，2003）以及*Pentaho Solution*（Wiley，2009）一书。

即使本书不涉及数据仓库和数据集市的设计，但为了有一个高质量的BI解决方案，你应该做的第一件事就是要花时间设计一个数据仓库模型。其他要做的事情还包括以下一些。

- **同行专家评审：**在项目开发过程中，要定期组织同行的专家评审以确保解决方案从一开始就符合标准。至少，在开始实现之前，你要检查ETL的设计。尽管这样会增加整个项目的成本，但可以降低项目返工的风险，而且能在效率和质量之间取得平衡。
- **标准化：**在项目开始前，先开发一些标准或模板，可以使项目更有可预测性。尽管建立这些标准或模板会花一些时间，但这些工作会节省以后的反复修改工作。

下面两部分将讲述其他两种好习惯，数据映射和命名规则（是上述标准化的一部分）。然后我们还总结了一些坏习惯，要尽量避免这些坏习惯。

数据映射

假设你的数据仓库已经设计好，你面临的第一个挑战就是数据仓库里的这些数据应该从哪里找到。第6章介绍了数据特征和抽取工作。只要源和目的都知道了，有趣的工作就要开始了。如何把源和目的映射上？需要什么转换？

首先，我们要找到源系统的元数据描述文档，或者至少能访问到元数据。如果有很多表和列，你又不想手工输入它们。一个好的方法就是创建一个表格，所有的目标列名放在一边，按照所属的表分组。然后为每个目标列定义数据源列。如果一个目标列来源于多个数据源列，也可以在表格里在增加一行或多行。无论是源列还是目标列，列的数据类型也要指定。然后在源列和目标列之间增加新的列，里面保存从源到目标列要做的转换。如果目标列没有对应的源列（例如，维度的版本号和代理键），指定一个可以生成对应值的某个过程，函数或Kettle步骤。

提示：几乎所有的数据库系统都能让你查到这个系统的表或列的元数据。Kettle可以通过表输入步骤读到这些元数据，并把它们写到其他地方。看如图11-2所示的例子，使用表输入步骤，在表输入步骤里写查询表sakila_dwh元数据的SQL语句，然后把这些元数据导出到Excel里。

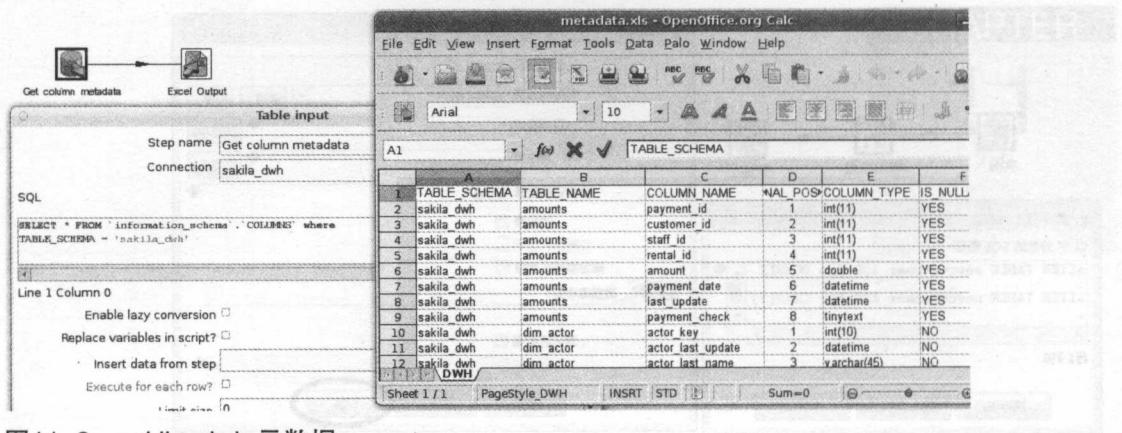


图11-2 sakila_dwh 元数据

名称和注释的规则

命名规则，也有可能出现两种极端的情况，一种是把命名规则设计得过于烦琐，一种是根本没有命名规则。如果你的团队多于一个人，或者你要把你的工作提交给第三方，合理的命名规则可以在很大程度上提高项目的可维护性。Kettle 里的每个对象都可以有一个名字，再结合作业、转换或步骤的图标，就更容易理解和记忆。

首先要记住不能使用相同的转换或作业的名字，如果已经使用了load_data 作为转换名，再使用 load_data 作为作业名就会引起混淆。如果使用jb_作为所有作业名的前缀，使用tr_作为所有转换名的前缀，则是一个好方法。另外，作业或转换实现的功能又或者操作的表名也可以作为文件名的一部分。如一个从customer 表中抽取数据的转换，可以命名为tr_e_customer, 如用于数据缓冲（staging）阶段的转换，转换可以命名为tr_stg_customer。加载到某个维度表的转换可以命名为tr_dim_customer等。另外还可以使用抽取数据的数据源作为转换名称的前缀。例如，若你从SAP系统里或从Siebel CRM系统里读取数据，转换可以命名为tr_stg_sap_customer和tr_stg_crm_customer。

给步骤起名字，只要能反映出这个步骤的功能即可。最后要强调的是注释，可以给转换或作业增加注释，这对于理解转换的作用非常重要。除了作业或转换的注释，还可以使用作业、转换、步骤、作业项的描述字段。和注释不同的是，这些描述可以和步骤等对象一起被复制粘贴。

Kettle例子里的GetXMLData-Different Options.ktr 转换是一个有注释的较好的例子。

在转换里，列名通常的命名规则是：对只参与转换和运算，而不输出的列，其列名应该以t_或tmp_ 为前缀。尤其在表的字段很多的情况下，这种命名可以方便地区分哪个列是临时使用的，哪个列是要输出的。使用“字段选择”可以容易地清除下面步骤不再使用的字段。

易犯的错误

除了“一上来就编码”这样的坏习惯外，下面还有几个开发人员易犯的错误。

- 不和最终用户交流：作为ETL开发人员，你可能认为和用户交流是项目管理人员的事情，或是开发前端页面的开发人员的事情。并非如此，只有业务人员才能告诉你，你交付的数据是否正确。业务人员还可以告诉你源系统中业务数据的含义（可能和文档中有

一些区别)。本章后面的“敏捷开发”部分讲述如何和业务人员结伴快速开发。

- **基于其他项目的经验假设：**可能你上一个项目的用户有一套比较好的备份系统，用户可以允许ETL工作的时间窗口为12个小时。但这不是你现在项目的工作环境。
- **忽略变化的需求：**在很多情况下，开发人员是在需求确定的情况下进行开发的，而在项目周期里很少考虑需求的变化。本章后面的敏捷开发部分将引导你解决这个问题。
- **忽略生产环节的需求：**一般ETL开发都是在开发环境中，使用的测试数据有限。在项目的前期，尽早使用生产环境的数据和硬件设备来检验你的工作。通常生产环境的机器性能更好，但是你可能会发现，在测试环境中30分钟就能完成的作业，在生产环境中要几个小时才能完成。
- **过于追求完美：**和其他ETL工具相比，Kettle已经可以节省很多时间，而且肯定也有其他方法，会做得比你现在的更好。但不要太追求完美；如果作业可以在规定的时间窗口内完成，有足够的错误处理机制，这样的作业就很好了，尽管不一定完美。额外的工作不一定会带来额外的价值。
- **没有测试：**如果作业可以运行在开发环境中，可以直接移到生产环境中吗？不行，技术上没有错误并不意味着结果是正确的。本章后面将会讲述。
- **不备份数据：**要定期备份数据，尽管资源库崩溃的可能性很小。

11.1.2 ETL流设计

ETL解决方案实际就是一个整体的过程。这个过程可以根据一定时间间隔触发，或者根据某些事件触发，如某个文件是否可用等。这个过程需要外部的输入，包括文件、记录和消息等，还需要转换来操作这些输入。简单地说，这个流程就像日常业务处理流程一样。所以我们要像处理业务流程图一样去处理一个ETL流程。所以我们为什么不用Kettle本身来处理这些流程？考虑如图11-3所示的例子。

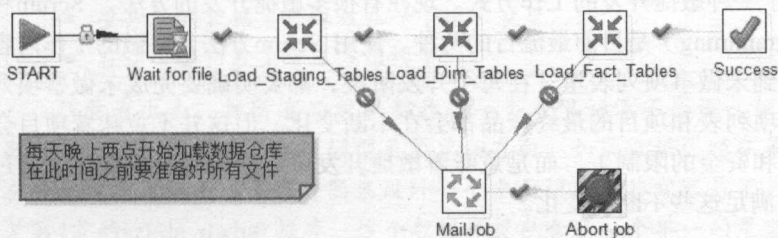


图11-3 作业设计

图11-3 显示了一个典型的作业流，这个作业流启动了其他作业和转换。这个图显示了业务流程，可以给业务人员使用，做好这个图只要几分钟的时间。做这个总体流程图的时候，我们并不需要具体实现图里的每个作业或转换的，只要有个符号，来说明某个环节的作用即可，这样可以和业务人员或行业专家讨论，这个图里还缺了什么。另外不要忘了写注释——计划在每天凌晨2:00启动作业。因为若计划启动时间和系统其他时间，如备份时间，冲突，那么需要的某些外部资源可能无法获取到，这个要在开发前提前讨论，以避免以后的争论。

11.1.3 可重用性和可维护性

结构化编程的一个重要原则就是使代码尽量具有通用性，可重用。Kettle的流程设计也遵循

这一原则。构建好的作业和转换都可以复用，通过转换和作业都可以嵌入到其他作业中，转换还能通过映射功能嵌入到其他转换中，这些都体现了复用的原则。第4章里有一个很好的复用的例子，`fetch_address` 转换通过映射功能嵌入到了`dim_customer`和`dim_store` 转换里。

通过使用变量也能体现复用的原则，尤其是重复的一些元素，如电子邮件地址、数据库连接等。Kettle 里有两种复用方式“复制/粘贴”和真正的复用。当你发现你在复制/粘贴时，就要考虑是否有必要把复制的对象放在一个转换或作业里，或者把复制的对象直接映射到其他处理流程中。使用“邮件”作业项插件会遇到的一个麻烦就是：要输入大量的参数（发件人、收件人、消息、SMTP服务器，等等）。所以我们一旦输入完一个就很容易把这个作业项到处复制。实际上只有在这些输入参数都使用了变量的情况下，你才可以这么做。如果用了变量，复制多少份都没有关系，变量的值只需要改变一次就可以——除非你想改变变量名。在做步骤/转换/作业的复制操作时你要问自己一个问题，如果改了要复制对象里设置的值，需要改动几个地方？如果答案是大于1个，那你就要想其他办法了。

回到前面说到了邮件作业项。Kettle 里有两个发送邮件的组件：一个在作业里，一个在转换里。这里我们讨论的是作业里的发送邮件功能。如果想在作业流程中，一有错误就发一个邮件，可以创建一个`jb_ErrorMail` 作业，里面就一个发送邮件作业项。`jb_ErrorMail` 作业可以放在任何你想发邮件的地方。如果需要改动邮件参数等，只需要改一次。

另一个复用的例子就是给数据库的连接参数使用变量。这样可以很简单地从开发环境移植到测试、确认、生产环境。在第13章中介绍移植。

11.2 敏捷开发

在第1章中，我们讲了一些敏捷开发的内容，这里再详细介绍一下。Pentaho和Kettle不只是敏捷BI的解决方案，更提供了一种敏捷开发的工作方式。现在有很多敏捷开发的方法，Scrum和极限编程（XP, Extreme Programming）是目前最流行的两种。使用Scrum方法，要做的工作内容（如代码块或转换）都要贴到未做事项列表里。在每个开发阶段，都要明确要完成未做事项列表里的哪几个事项。未做事项列表和项目的最终产品都会在不断变化。但这并不意味着项目会无休止地开发下去（有时间和资金的限制），而是意味着敏捷开发可以适应不断的变化。通过ETL设计工具（Spoon）可以满足这些不断的变化。

从Kettle 4 开始，Spoon成为Pentaho BI 开发套件中的基石。以前Spoon只是一个单纯的ETL开发工具。现在它提供了不同视图功能，还有建模和可视化视图。这些新的特性，使Spoon可以设计多维模型并使数据可视化。

Spoon从Eclipse里借鉴了视图的概念，这样可以处理有相似接口的不同种类的流程。社区版有数据整合视图、模型和可视化视图（模型和可视化视图通过插件提供——译者注）。企业版除了社区版提供的视图外，还有调度视图。敏捷BI的工作流程如图11-4所示，这里有三个环节：ETL设计、建模和数据可视化。

数据整合环节我们已经比较熟悉了，但建模和可视化是Kettle 4 新加的功能。把这几个相关的视图都集合在一起，放在一个工具里。这样开发人员就可以和业务人员一起开发，把数据展现给业务人员，告诉业务人员将来的数据是什么样子的。但是要注意，这些敏捷开发方法适合开发原型，还不能用来设计一个企业级的数据仓库。

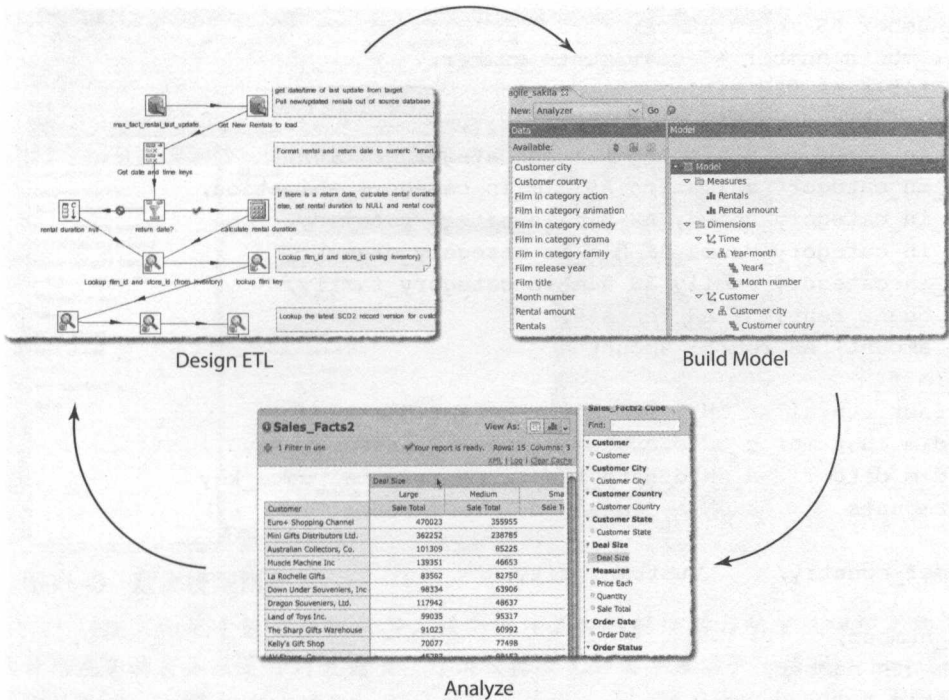


图11-4 敏捷BI

说明：在编写本书时，Kettle的模型和可视化功能还限制在输出表里。但对于原型开发并不算障碍。

例子：使用 Kettle进行敏捷BI开发

为了在Spoon里快速分析数据仓库里的数据，要做两件事情：

- 要分析的数据集可以放在一个输出表里。
- 在这个输出表上建立多维模型。

尽管看上去，把所有的数据都放在一个表里好像不合适，但随后可以看到，这没有问题。事实上，这样反而会使模型更简单和直白。如果数据已经在星型模型数据库里了，那么在模型视图里使用模型设计器来设计一个平面模型就非常容易了。这个例子使用的数据库还是第4章的sakila_dwh数据库。这个数据库里包含了一个单一的星型模型，这个星型模型里有电影、顾客、商店、演员等维度表，还有租赁业务的事实表。你需要一个“反正规化（行转列）”步骤来创建一个把维度表和事实表组织在一起的大视图。

这样的平面数据模型还有一个名字叫：单属性集接口（One Attribute Set Interface, OASI）模型，荷兰的数据仓库专家 Harm van der Lek 博士在他的*Sterren en Dimensies (Stars and Dimensions)* 书里描述了这种模型。这本书（ISBN 90-74562-07-8）可以在网上找到（<http://array.nl/Boeken>）。单属性集接口模型里包含了星型模型里所有维度表和事实表的非关键字属性。下面的SQL查询也相当于一个“反正规化”步骤，来生成OASI模型。

```
SELECT
    c.customer_country AS customer_country,
    c.customer_city AS customer_city,
    d.year4 AS year4,
```



```
d.month_number AS month_number,
  d.year_month_number AS year_month_number,
  f.film_title AS film_title,
  f.film_release_year AS film_release_year,
  f.film_in_category_action AS film_in_category_action,
  f.film_in_category_animation AS film_in_category_animation,
  f.film_in_category_comedy AS film_in_category_comedy,
  f.film_in_category_drama AS film_in_category_drama,
  f.film_in_category_family AS film_in_category_family,
  sum(r.count_rentals) AS rentals,
  sum(a.amount) AS rental_amount
FROM dim_film f
INNER JOIN fact_rental r ON f.film_key = r.film_key
INNER JOIN dim_customer c ON c.customer_key = r.customer_key
INNER JOIN dim_date d ON d.date_key = r.rental_date_key
INNER JOIN amounts a ON a.rental_id = r.rental_id
GROUP BY
  customer_country, customer_city,
  year4,
  month_number,
  year_month_number,
  film_title,
  film_release_year,
  film_in_category_action,
  film_in_category_animation,
  film_in_category_comedy,
  film_in_category_drama,
  film_in_category_family
```

注意这并不是全部非主键属性，这只是一个子集，用来演示模型/可视化概念的原理。在ETL这个阶段的数据分析的主要目的是弄清用户想要什么样的数据，所以不必要把所有的数据都加载到目标表里。使用 Kettle的样本步骤可以从一个大表里随机抽取指定行数的数据行。在如图11-5所示的例子中，我们随机读取 1000 行数据加载到目标表里。

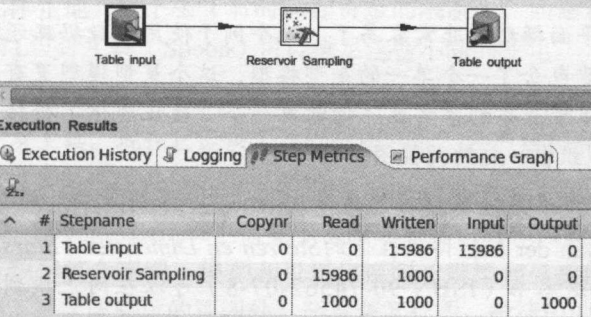


图11-5 使用样本步骤加载数据

运行了上面这个转换后，数据库里有了数据，才能使用Kettle自带的建模工具。右键单击“表输出”步骤，右键菜单底部就是“建模/可视化”选项。也可以单击工具条上的“建模”按钮。建模窗口打开后，其中有三个窗格，从左到右分别是“可用的数据项”、“模型”、“模型里各节点的属性”。图11-6是上面的“表输出”步骤上模型的例子。

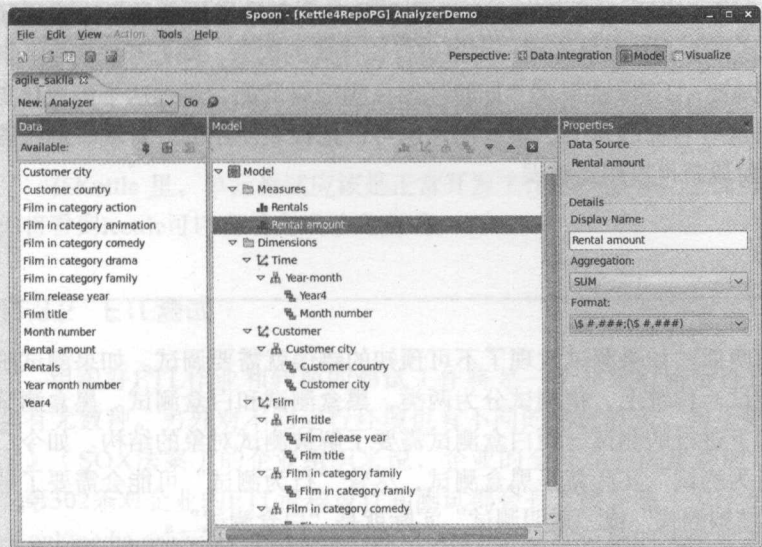


图11-6 数据模型例子

创建上面这个模型非常简单，模型设计器会根据规则自动生成模型里的名字。模型设计器左侧的窗格里是可用数据项，可用数据项里如果字段名带下横线，下横线会自动被一个空格替代，如果没有下横线，还使用原来的字段名。这种名称的变化是自动完成的，不需要其他元数据模型工具。建立这个模型时要注意类似“Film in category drama”或“Film in category comedy”这样的属性，因为这些属性之间没有依赖关系，所以要把这些属性放在不同的层次（Hierarchy）下面。另外一些属性如Year和Month，Country和City，它们有依赖层次关系，这些属性应放在同一个层次下面。通过图11-7可视化视图里的View→by Category可以快速查看每个层次下的对象。我们也可以明白为什么这个工具也可被称为Pentaho 分析器：我们只要单击几下鼠标，就可以查看利润最高的top 10 国家，也可以专门分析喜剧电影，看哪个国家的得分最高。

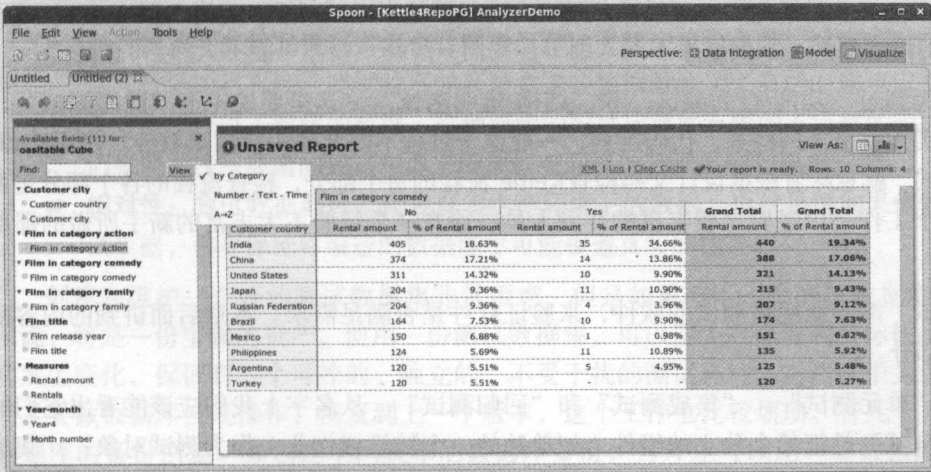


图11-7 分析报告的例子

这里还有一个功能，就是“报表向导”，可以用来创建标准的报表。报表向导需要元数据层，元数据层需要先发布到Pentaho服务器，元数据层也可以在门户里使用。这里我们不做太多说明。

不过敏捷BI不是解决一切技术问题的万能药。它只是开发人员和最终用户的一个沟通桥梁。利用这个可视化的ETL工具开发人员和最终用户可以进行迭代式的沟通,降低开发成本。例如,开发人员如果不明白“成本”、“价格”、“日期”等一些行业词汇,通过这个可视化工具生成视图后,可以快速明白这些名称的含义。最终用户也可以把报表使用的“订单日期”改成“交付日期”,这样在开发阶段就可以纠正错误。

11.3 测试和调试

测试和调试是一个事情的两面。如果测试发现了不可预知的错误就需要调试。如果测试阶段运行正常,也可以不需要调试。传统上,把测试分为两类,黑盒测试和白盒测试。黑盒测试是在不了解被测试对象的结构下进行的测试,而白盒测试需要了解被测试对象的结构。如今广泛使用的“功能测试”或“行为测试”实际就是黑盒测试,尽管“行为测试”可能会需要了解一些被测试对象的结构。而“结构测试”或“透明测试”实际就是“白盒测试”。

这些测试方法都是用来测试软件的,要求被测试软件提供了用户接口和测试用例。例如,一个订单系统,功能测试要求每个客户都要有一个电话号码,如果没有电话号码应该有警告提示。测试用例就是保存一个没有电话号码的客户信息。正确的结果应该显示一个错误信息,记录不能够被保存。如果最后的测试结果提示了错误信息,记录也没有被保存,则测试成功,否则测试失败。功能测试并不关心系统如何校验电话字段、如何处理异常。

而ETL过程则不同,它没有最终用户界面,也不是由用户来触发的,一般是通过调度自动完成的。和普通软件的功能测试相比,ETL的“功能测试”有自己的特点。ETL的“结构测试”也和普通的“结构测试”不同,甚至更简单:可以使用同一个工具来做开发和测试。我们后面会看到,Spoon 提供了很多方法便于ETL流程的测试。

11.3.1 测试活动

关于测试的内容很多,我们这里不深入细节,我们只解释一些你可能不太熟悉的概念。

我们曾经提到过“功能/行为测试”和“结构/透明测试”。这里要说一下“静态测试”,“静态测试”实际就是文档检查,代码走查,一般这个过程不需要使用被测试软件。使用Kettle时,“静态测试”就意味着根据设计文档检查Kettle 流程的每个部分,检查流程的各个部分是否正确。通常这个工作是由经验比较丰富的开发人员,检查一个经验不太丰富的新手所做的那部分工作。

“动态测试”需要使用被测试的软件,来验证软件是否满足需求。本章后面讲到的内容都属于动态测试部分。

另外还有“单元测试”、“集成测试”和“回归测试”。从名字上我们应该能看出它们的含义:“单元测试”是把某个独立的组件(如单独的一个转换或作业)作为测试对象;“集成测试”是对完整解决方案的测试,通常是包括了很多作业和转换;“回归测试”是某个组件发生了变化后的测试,保证整个解决方案在某个组件变化后,仍可以正常工作。

最后的测试称为UA或“用户确认测试”。UA测试对ETL来说好像并不需要,因为最终用户并不需要和ETL系统直接打交道。用户只和通过ETL流程加载到数据仓库里的数据打交道。用户

只需测试最终数据的完整性和正确性。一般，都是最终用户可以找到最终数据计算上的错误或发现某些数据丢失。数据UA测试是BI解决方案UA测试的一部分，除了数据部分，BI解决方案UA测试还包括前台报表和分析工具等测试。BI解决方案UA测试超出了本书的范围，本书我们只讨论ETL测试。

在Kettle里，单元测试应该是正常开发工作的一部分。下面部分将会重点介绍这部分测试，你将看到Kettle可以自动完成单元测试。

11.3.2 ETL测试

组织好ETL作业和转换的测试工作将是一个非常具有挑战性的工作，因为可能的测试用例会有无数种。另外对不同运行环境也有不同的测试需求。对于需要遵守《萨班斯-奥克斯利法案》（SOX法案）的企业组织来说，企业的全部业务流程都应该是可以监控的。特别是法案的第302条对企业的ETL流程设计和测试都会有直接影响（关于法案的进一步信息可参考http://en.wikipedia.org/wiki/Sarbanes-Oxley_Act）。这些苛刻的法律要求都需要ETL解决方案要百分之百的正确和可测试。但事实上，你会发现如果要测试所有异常情况几乎是不可能的。我们只能保证，在通常的环境下，可以正确而完整地转换一组给定的数据集。请注意这里的用词，“通常的环境”和“一组给定的数据集”。尤其后面这个词，是一组给定的，而不是任意一组。

测试数据需求

要测试ETL流程，就要有一组测试数据。这组测试数据在数量和内容上越接近生产环境数据越好。有时候很难拿到类似生产环境的数据，有时候可能还有法律方面的障碍。在准备测试数据时我们要考虑下面几个方面。

- **数量**：项目初期的单元测试不需要完整的测试数据。但在项目进行到某个时间点时，就要使用一套完整的生产环境数据做集成和压力测试。在单元测试阶段，只要使用一定数量的测试数据，最好使用生产数据的随机样本数据作为测试数据。
- **私有性**：生产数据里有类似会计账户信息、医疗记录信息等一些敏感的私有信息，因为这种数据的私有性，测试时不能直接拿这些数据做测试。有的时候可以拿到从这些系统里导出的经过了混淆的数据，这种混淆的数据比那些有限的不完整的数据集要好。但我们要保证这些混淆的数据和原来的数据有同样的数据类型和数据长度。
- **相对性**：测试数据要可以代表真实生产环境下的数据，也就是说如果你需要手工编造测试数据，你要保证你编造的数据能尽可能覆盖真实的场景。

创建和维护一个好的测试数据集非常困难，但是如果有了一个测试数据集，对ETL项目来说，将是一份宝贵的资产。使用一份测试数据集，可以避免数据源和目标数据库里的数据经常发生变化，保证有一个可控的、独立的、不受干扰的测试环境。在测试中为保证测试环境还经常要做数据库恢复操作，恢复到上一个版本，这个工作也比较烦琐。有几个方法可以做这种数据库的备份恢复：如创建不同版本的数据库备份，另外还可以用一些数据库测试工具，如DbUnit（<http://www.dbunit.org>），这是一个JUnit测试框架的扩展。

完整性测试

测试工作的第一个目标就是要测试数据的完整性，要确保所有的数据都被处理。下面有几

个测试数据完整性的方法。

- **计算记录数**: 计算读、写、转换、更新、拒绝的总行数。如果使用Kettle, 这是一件很容易的事情, 因为运行转换时, 这些都是转换的度量数据。
- **哈希合计**: 分别在源系统和目标系统中计算某个关键列的合计值或哈希合计值。例如, 可以使用销售数量合计来验证源系统和数据仓库中的数据个数是否一样。
- **校验值**: Kettle可以计算一列或多列的校验值。通过比较校验值来验证行数据的完整性。

上面的这些验证数据可以在转换的过程中生成, 但最好的方法还是把它们保存在数据库里。这样便于生成关于这些表或ETL的报告, 并记录下ETL活动的过程。我们拿第4章的load_rentals 转换做个例子。如果要测试这个转换的完整性, 有下面几个问题:

- 最初始的加载工作是否正确, 是否从源系统中加载了所有的记录?
- CDC选项是否工作正常, 是否没有记录遗漏?
- 抽取到的记录是否都加载到了目标表里?

为了使测试自动运行, 我们需要捕获转换每次运行时写到日志表里的记录数。在14章会讲到日志和审计, 这里涉及日志的一些概念。

1. 首先, 打开load_fact_rentals 转换, 使用Ctrl+T或菜单项的“编辑”→“设置”打开转换属性对话框。在“日志”标签下, 可以定义几个日志选项, 为了简化我们只设置转换日志。
2. 建立日志数据库连接。对这个例子而言, 我们使用了sakila_dwh 库作为日志数据库(在实际环境中, 日志库和业务库应该是分离的。)
3. 在“日志表”文本框里输入日志表的名字, 并单击SQL 按钮。创建日志表的脚本会自动生成, 直接执行这个脚本。执行脚本之前确认所有需要的日志字段都已被选中。

说明: 如果想修改日志表的字段, 还要使用SQL 按钮来生成修改表的SQL 语句。否则转换/作业就会报告错误。

4. 在“日志字段”列表的“步骤名称”列里设置你要监控哪个步骤的度量值。如图11-8所示, 因为我们只关心转换的总数, 所以我们只要把转换的第一个步骤和最后一个步骤设置到步骤名字段里。

如果对这个例子做完整性测试, 就要验证下面的几个测试用例:

1. 在第一次初始化时, 源系统中的所有数据都被加载到目标数据库里。
2. CDC 机制要正确地识别出在第一次初始化后源系统里新增加的数据。
3. CDC 机制要正确地识别出在第一次初始化后源系统里更新的数据。
4. 源系统里新增加的数据要正确插入到新系统中。
5. 源系统里更新的数据, 要相应更新目标表里的数据。

执行这些测试用例非常简单: 对第一个测试用例来说, 先在一个空的fact_rental 表里做数据加载, 再和kettlelog 表里的读数据的总数做比较。第二个测试用例要求往 rentals 表里增加一行。第三个测试用例要求更新 rentals 表里的一行。第四个和第五个测试用例测试增加和更新的数据是否都同步到目标表里。

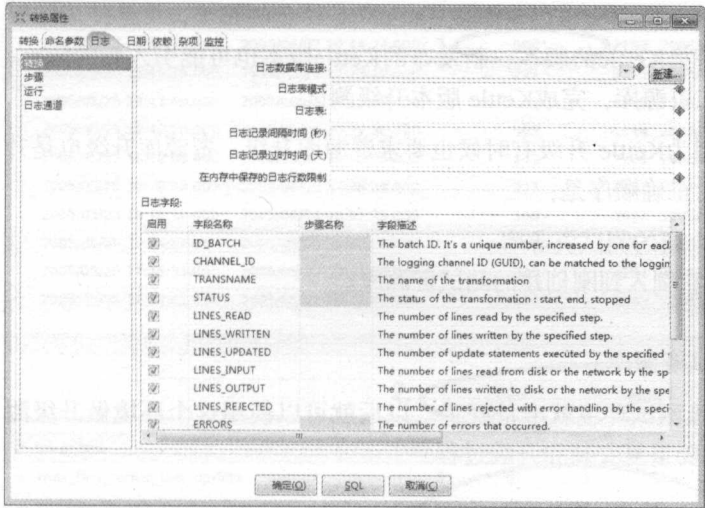


图11-8 转换日志表设置

测试数据转换

在数据从源系统中读出来后，再根据你设计的流程处理，实际上数据就发生了这样或那样的转换。不管逻辑和处理流程怎样变化，总是有一件事情是可以（也是应该）测试的：对给定一个输入的期望输出。在理想情况下，你有一组使用场景，以及针对每个场景的输入项和期望的输出项。尽量避免手工执行测试，因为即使只有一两个源系统的流程，也可能会有很多种测试用例。

用来保存“使用场景”、“输入”、“期望输出”最常用的工具就是无处不在的电子表格。Kettle可以从XLS这种电子表格中轻松地读出数据，但对于新转换的初始阶段测试还有一种更好的方法：使用“自定义常量数据”步骤（Data Grid），使用这个步骤可以保存要输入数据的类型和数据，以及期望的输出，这样可以直观看到要测试的数据和期望结果。

警告：使用“自定义常量数据”步骤来检查转换结果是否正确，尽管很方便，但也只能用于测试的第一步。对于后面的测试，还是要使用普通的输入步骤来代替它。

我们继续看 load_rentals 转换，这个转换还有很多要测试的地方。如一个商店在商店的维度表中没有怎么办？Kettle是否有办法处理这种情况？我们检查所有的“数据库查询”步骤，可以看到所有的“数据库查询”步骤都选中了“查询失败则忽略”选项。另外还有一个计算步骤用来计算“rental_duration”，这个步骤是否能得到正确的结果？如果归还日期是空的怎么办？如果归还日期小于出租日期怎么办？一般都要通过数据校验或错误处理来解决这些问题。

测试自动化和持续集成

测试应该自动运行，建好一套测试用例后，就应该定期地跑这个测试。可以通过自动调度作业来实现自动测试。另外无论什么时候新增加或修改一个作业，都要把它们加入到自动测试总作业中，以实现持续集成。

升级测试

Kettle 发展很快，每一两个月就会发布一个新版本。新发布的版本一般都会修订一些bug，

引入一些新的功能，提高性能，等等。所以应该尽量跟随Kettle产品发布的脚步，尽量使用新版本。当然在升级Kettle 时不能只是替换Kettle的版本，新发布的Kettle版本也可能引入新的bug，所以要有一个独立的测试环境和测试资源库，完成Kettle 版本升级测试。

之所以需要独立的资源库，是因为Kettle 升级有时候也要求资源库升级。资源库升级也是升级测试的一部分。建立独立资源库的正确顺序是：

- 使用Kettle 当前版本创建一个新的测试资源库。
- 导出当前生产资源库，并把它倒入到刚创建的测试资源库。
- 安装新的Kettle 版本。
- 使用新的Kettle 版本升级测试资源库。

上面过程都执行成功，你的升级测试环境就准备好了。以后就可以使用这个环境做升级测试。对于要升级的每个Kettle 版本都要重复安装和升级过程。

11.3.3 调试

调试是发现和修改软件错误的过程。在Kettle里，调试基本是由程序强制完成的，就是说很多错误其实在开发过程中就被解决了，如我们要允许一个复杂的转换，所有的数据库连接都要工作正常，数据可以访问到，可以处理异常和错误，数据可以写到目标数据库中。如果这个过程的一个环节错误，Kettle 会抛出一个异常告诉你错误，这些其实就是一种调试。Kettle的调试功能要使用工具条上的预览和调试选项。

通过预览和调试选项，可以执行转换到任意断点。图11-9显示了预览和调试这两个功能按钮。

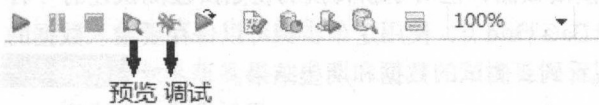


图11-9 预览和调试按钮

尽管这两个功能都打开同一个窗口，但它们之间有一点区别。如果打开的是调试窗口，可以看到所有的步骤都在左侧列出来，当前激活的步骤（使用预览和调试功能时，流程图中当前选中的步骤）被选中并高亮显示。窗口的右侧显示的是一个大的条件面板（类似于“过滤”步骤的条件面板）和右上方的两个选项。在单击“快速启动”按钮后，这些选项决定着“调试”窗口的执行方式。选中“获得前几行（预览）”选项，并设置“要获得的行数”输入框，Kettle就会在处理到指定的行数时停止，并显示出指定行数的数据。如果选中“满足条件时暂停转换”，并设置一个条件，Kettle在运行中如果满足了这个条件就暂停，并显示处理过的数据行。如果同时选中这两个选项，只有预览选项生效。

如果选中了“满足条件时暂停转换”选项，“要获得的行数”设置也可以生效。可以只获得满足条件时最近的N条记录，并以倒序排序。例如我们选中了“满足条件时暂停转换”选项，并设置获得的行数为10，Kettle 设置了断点的步骤在遇到满足条件的记录时停止运行，并显示进入了这个步骤的最近10条数据。图11-10的调试窗口显示了当触发条件是customer_id =500 时返回的最近 10行数据。

| ey | rental_datetime | return_datetime | customer_id | inventory_id | last_update |
|----|-------------------------|-------------------------|-------------|--------------|-------------------------|
| | 2005/05/25 18:57:24.000 | 2005/06/02 20:44:24.000 | 1. 500 | 2153 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 18:45:19.000 | 2005/05/28 17:18:19.000 | 227 | 3725 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 18:43:49.000 | 2005/06/03 18:13:49.000 | 19 | 4108 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 18:40:20.000 | 2005/05/29 20:39:20.000 | 503 | 1044 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 18:30:05.000 | 2005/05/30 19:40:05.000 | 242 | 3289 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 18:28:09.000 | 2005/06/03 22:46:09.000 | 317 | 2833 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 18:18:19.000 | 2005/06/04 00:01:19.000 | 196 | 3627 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 17:54:12.000 | 2005/05/30 12:03:12.000 | 108 | 794 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 17:46:33.000 | 2005/05/27 15:20:33.000 | 310 | 4281 | 2006/02/15 21:30:53.000 |
| | 2005/05/25 17:30:42.000 | 2005/06/03 22:36:42.000 | 384 | 3343 | 2006/02/15 21:30:53.000 |

| | | 2. Close Stop Get more rows | | | | | | |
|----|--|--|-------|---------|-------|--------|---------|---|
| # | Stepname | Copynr | Read | Written | Input | Output | Updated | R |
| 1 | max_fact_rental_last_update | 0 | 0 | 1 | 1 | 0 | 0 | |
| 2 | Get New Rentals to load | 0 | 1 | 16044 | 16044 | 0 | 0 | |
| 3 | Get date and time keys | 0 | 16044 | 16044 | 0 | 0 | 0 | |
| 4 | Lookup dim customer key | 0 | 423 | 422 | 601 | 0 | 0 | |
| 5 | Lookup dim staff key | 0 | 324 | 323 | 3 | 0 | 0 | |
| 6 | Lookup dim store key | 0 | 224 | 223 | 5 | 0 | 0 | |
| 7 | Lookup film_id and store_id (from inventory) | 0 | 593 | 592 | 589 | 0 | 0 | |
| 8 | lookup film key | 0 | 522 | 521 | 1000 | 0 | 0 | |
| 9 | return date? | 0 | 16044 | 16044 | 0 | 0 | 0 | |
| 10 | calculate rental duration | 0 | 9784 | 9783 | 0 | 0 | 0 | |

图11-10 使用断点

在这个图中，第一处圆圈是满足断点条件的记录。第二处圆圈是此时可以进行的操作：“关闭”、“停止”、“获得更多行”。第三处圆圈显示了此时已经处理的行数。关于第二处圆圈的几个操作，我们要注意几点：即使点击了“停止（stop）”按钮，也不能回滚到最初始的状态，因为Kettle可能已经把满足条件之前的数据加载到了目标表。“关闭（close）”按钮只会关闭当前窗口，不会停止转换，转换还在暂停状态，这可以从工具条上“暂停”和“停止”按钮的颜色看出来，如图11-11所示。

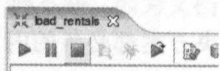


图11-11 处于暂停状态的转换

即使单击了“停止”按钮，在满足条件之前，数据可能已经加载到了目标数据库。如果不想把部分数据加载到目标数据库，可以在最后的加载步骤之前加一个“阻塞”步骤。例如fct_load_rentals这个转换，可以在“插入/更新”步骤之前加入这个“阻塞”步骤。

警告：不只是“调试”，“预览”窗口里的“停止”按钮也要注意同样的问题。即使选择的是“预览”，转换也开始运行了，如果转换里有删除步骤，也会真的删除数据库里的数据。

Kettle也支持逐行调试，但和其他工具的逐行调是不太一样的。像上面那样启动一个调试窗口，单击预览数据窗口里的“获取更多行”按钮，此时预览数据窗口就会关掉，当再单击工具条上“暂停”按钮时，预览数据窗口又会重新出现，里面只有一条记录。这个过程可以重复直到所有数据行预览完成。

Kettle 4 有很多新功能，在调试方面，钻取（drill-down）和嗅探（sniffing）功能非常有用。

你在开发的时候可能会发现，可以通过右键的方式打开作业里的子作业、转换，或通过右键打开转换里的映射（子转换）。现在，在运行阶段也能使用这种钻取功能了。另外在运行阶段，通过右键菜单还能看到有嗅探功能，嗅探功能可以实时看到一个步骤的输入行、输出行、错误行。可以同时查看多个步骤的嗅探，如图11-12所示。图11-12展示的是一个“生成记录”步骤和一个“Java脚本”步骤，把两个字段连接起来，并输出到文本文件。嗅探功能把“生成记录”步骤的输出行和“文本文件输出”步骤的输入行都显示出来了。

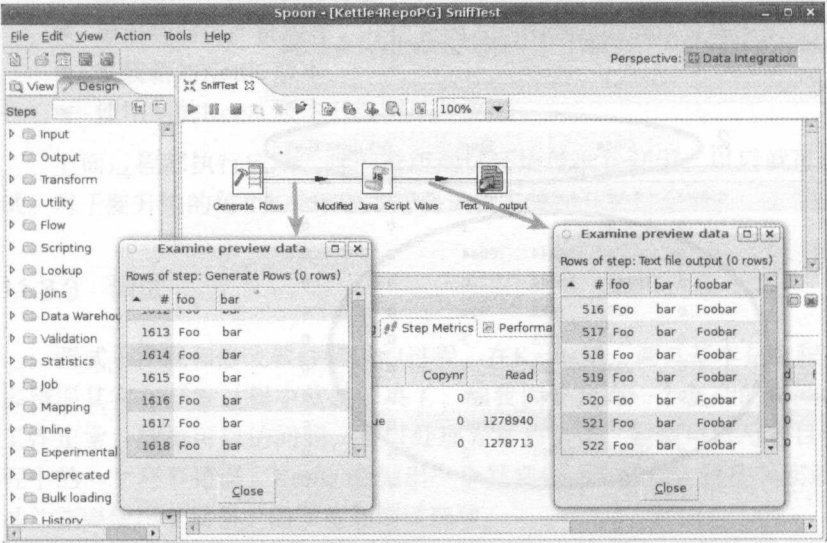


图11-12 嗅探一个正在运行的转换

Matt Casters在YouTube的视频上演示了嗅探功能（<http://www.youtube.com/watch?v=imvpQ8FFo-A>）。

11.4 解决方案文档化

在ETL开发里最重要的而且最容易忽略的就是写文档。文档是任何IT解决方案都必备的内容，ETL也不例外。写文档有很多目的。一个好的文档能在很多方面帮助我们，例如：

- 可以记录下ETL的需求，以及如何设计。
- 文档能快速地训练新的开发人员，帮助他们理解ETL的实现。
- 在一些情况下，文档可以理清数据之间的血统关系，帮助审计。

11.4.1 为什么实际情况下文档很少

尽管文档的作用非常大，但在实际项目里文档却非常少。因为开发人员也不喜欢写文档，更不喜欢维护文档，因为写和维护文档在短期之内都看不到收效，而且要赶项目的时间，不能延期，所以花在文档上的时间就很少。关于Kettle文档，我们经常可以听到下面一些说法：

- Kettle的转换和作业本来就是图形的，一些ETL开发人员认为，这种图形本身就可以解释流程的含义，所以不必写文档。
- 一些开发人员认为写文档实际是个坏毛病，因为文档都是会过时的，文档会给后来的人

留下错误的或不完整的信息。

■ 最后一个常听到的就是，文档不用写，因为根本没人看。

这些都是不想写文档的常见借口，除了ETL开发人员，其他各种开发人员（C/C++、数据库）都会有类似借口。这些借口看着有道理，但实际情况如下。

神话1：我的软件可以自我解释

典型的，说软件可以自我解释的人一般就是这个软件的开发者。对开发者来说软件可以自我解释，但对于后面的维护和修改人员来说，软件的逻辑就不是这么清晰了。

ETL的作业和转换也是一样，它们只能自我解释它们做了什么。但它们不能解释设计思路。在很多情况下，文档不仅仅要解释做了什么，更要解释为什么这么做。

神话2：文档和软件比，总是会过时的

文档的确会过时，但这不能否定文档的作用。如果文档过时太快，只能说明写文档时并没有考虑到软件的设计和开发方向。

如果开发人员和项目经理把文档作为开发过程的一部分，而且对文档也进行测试（如果测试软件一样），文档永远也不可能过期。就是说，文档应该作为开发的一部分，如果软件更新了，文档也应该同时更新。

神话3：没有人读文档

同样的原因，读文档的作用也常被低估，如同写文档的作用。读文档并不被认为是什么工作，所以也不是必需的。但公平来说，没有人读文档是因为没有可读的文档。即使有文档，这些文档可能也是被那些不相信有人去读的人写的（这样会影响文档的质量）。

这样就构成了一个糟糕的循环：因为写文档的人不相信有人去读，所以文档质量不高，写得不好。因为文档写得不好，没有人喜欢读。因为没有人去读，下次就不会有人再去写了。

11.4.2 Kettle的文档功能

一般我们写ETL解决方案的文档都使用一些工具，如字处理器、wiki页面或静态页面。这些方式的优点在于可以快速发布，在计算机屏幕上阅读或者打印出来。

这些方式的缺点在于，写文档和维护文档的过程脱离了ETL开发过程，变成了另一个流程，这样加大了文档和ETL开发之间的距离。在理想情况下，写文档也应该是开发过程的一部分。写文档应该是一个很自然的过程，无论什么时候改变了ETL流程，这种改变应该随时体现到文档中。这样文档就是最新和最准确的。

因此产生ETL文档的过程应该尽可能简单。不能让开发人员再开发一个应用来随时记录ETL的变化。

Kettle 提供了几个功能，可以在作业和转换里嵌入供人阅读的信息。开发人员可以利用这些功能来记录ETL方案设计的思路 and 原因。当前Kettle 还不支持把这些嵌入的信息转换成文档。但在本章的后面部分，可以演示怎样把这些信息抽取出来生成文档。

Kettle的文档功能如下。

- **描述性的名称：**一般在给作业、转换、步骤、作业项、字段等命名时，可以使用业务描述性的名称，这没有什么限制。只要这些名称在它们的作用域内不重复。例如，在一个转换里，步骤的名称不能重复，除此之外没有其他限制。
- **注释：**注释是放在作业或转换里小的文字面板。它可以包含任意文字，可以用来强调作业和转换的一些特征。在 Spoon 里，可以通过右键菜单来创建新的注释，另外还可以设置注释的字体和颜色。
- **在转换和作业设置里的描述字段：**在转换或作业的设计画布上右键可以打开转换或作业的设置窗口。在设置窗口里有输入项用来描述作业或转移。“描述”和“扩展描述”字段用来输入作业或转换的文字描述。“状态”字段用来输入作业或转换的状态，可用的状态有“草案”和“产品”。版本字段用来输入一个版本号。作业属性对话框如图11-13所示。
- **参数：**在作业或转换的设置对话框里还可以编辑参数。参数也支持描述字段，通过描述字段，可以记录关于这个参数的一些描述信息，例如参数的目的、参数期望的数据类型、数据格式、一些参数的例子，等等。

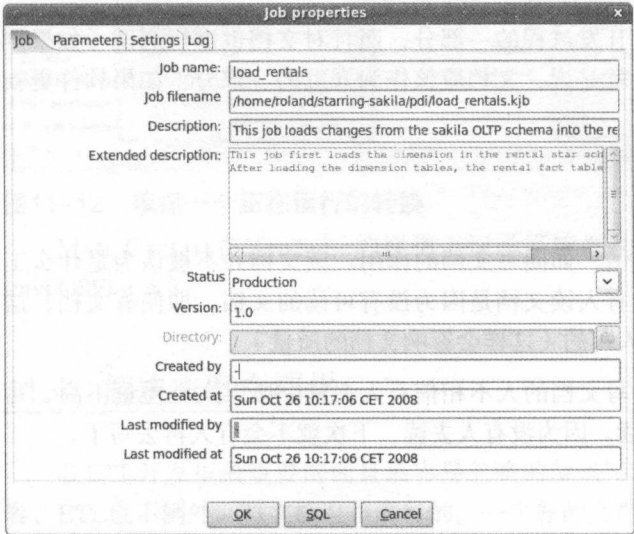


图11-13 使用描述字段将作业文档化

尽管上面这些功能非常有限，但至少可以让你记录下正在开发的作业或转换的必要信息。因为这些描述信息和作业或转换保存在了一起，所以当你开发和维护ETL作业时可以方便地编辑它们。

Kettle还有两个重要的文档功能没有实现。首先需要有一个规则，来规范如何使用Kettle的文档功能。其次Kettle 文档功能提供的这些描述信息只是Kettle 文档化的前提条件，而并不是文档本身。在本章后面部分，我们将看到如何抽取转换和作业里的描述信息来实现真正的可供人阅读的文档。

11.4.3 生成文档

当前Kettle不提供生成文档的功能。在编写本书时，Pentaho 正在开发一个自动文档的功

能。但是这个功能可能只用于Kettle企业版（自动文档功能在Kettle 4.3以后的社区版也可以使用——译者注）。

另外还有一个由 Roland Bouman（本书的作者之一）开发的社区项目，这个项目可以把一组作业或转换生成HTML。这个项目称为 kettle-cookbook，在Google code 上可以找到这个项目：<http://code.google.com/p/kettle-cookbook>。

kettle-cookbook 使用了几个Kettle 作业和转换来递归查找一个目录下的所有.kjb和.ktr文件。然后使用XSLT 样式转换文件将XML 转换成 HTML。人们就可以从浏览器里查看到这些文件。最后 kettle-cookbook 还生成了一个用于索引的目录，用于导航作业和转换。

要使用kettle-cookbook生成文档，需要按照下面的步骤进行：

- 把所有作业和转换放在一个目录下。这个目录就是“输入目录”。可以使用Kettle的导出功能，导出功能可以分析作业和转换的依赖关系，把依赖的转换或作业也导出来。Kettle的导出功能在“主菜单→导出→导出链接的资源……”。也可以导出在资源库中的作业和转换。
- 创建一个空目录，作为“输出目录”。
- 运行 document-all 作业。这个作业位于kettle-cookbook的pdi目录下。这个作业需要两个参数，INPUT_DIR 和OUTPUT_DIR，分别设置为输入目录和输出目录。

当 document-all 作业运行完成后，输出目录就会包含生成的HTML文档。此时可以在浏览器打开index.html文件，查看文档。kettle-cookbook 项目的网站演示了输出的例子。

11.5 小结

本书前面部分主要讲解的是构建ETL解决方案的基本概念。本章是本书第三部分“管理和部署”的开始章节，主要对ETL开发生命周期进行了概括性的介绍，主要涵盖了下面几个主题，这几个主题也是一个成功ETL项目的基础。

- **解决方案设计：**我们知道了ETL设计中的一些最佳实践和糟糕的实践。知道了数据映射和命名规则的重要性，我们列出了ETL项目中几个常见的错误。最后我们介绍了ETL流设计和可重用性。
- **敏捷开发：**了解了Kettle如何支持敏捷BI项目。通过例子学习了Kettle的建模和可视化功能。
- **测试和调试：**我们明白了在ETL里，测试是什么意思，以及测试面临的挑战。测试和调试的最后部分告诉你如何在Kettle 里使用调试工具来检查数据。
- **文档：**我们了解了文档的重要性，以及Kettle通过嵌入到转换或作业里的一些描述信息而提供的文档功能。最后我们学习了如何通过kettle-cookbook生成可以供人阅读的HTML格式的文档。

第12章 调度和监控

在本章我们主要讲述Kettle如何运行在生产环境中。在生产环境中，ETL任务一般都是以固定时间间隔来运行的，给数据仓库或其他系统提供数据。这种自动定期执行的任务就是调度。用于定义和管理这些定期执行的任务的程序就是调度程序。调度和调度程序是本章第一部分的主题。

除了调度以外，在生产环境中还需要监控调度过程和运行的结果，以便运维人员可以快速定位以及修复错误。例如需要一定的通知机制，来通知自动调度是否正常运行。另外，要采集运行中的数据，来判断运行情况。这样的程序我们称为监控程序。在本章的第二部分我们讨论监控Kettle作业的几种方法。

12.1 调度

本节我们讨论如下两种调度方法。

- **操作系统级调度：**调度是一种通用的需求，所以一般操作系统会提供标准的调度方式。例如类UNIX系统的cron命令和Windows系统的计划任务。这些调度程序可以调度任意的程序，也可以调度Kettle的命令行。
- **内置在Pentaho BI Server里的Quartz调度：**Kettle是Pentaho BI组件的一员，很多Kettle用户也使用Pentaho的BI Server。内置在Pentaho BI Server里的Quartz调度程序可以执行一个包含有Kettle作业的动作序列（action sequence是Pentaho BI Server的工作流文件。——译者注）。

12.1.1 操作系统级调度

所有操作系统级的调度都基于命令行：通过shell脚本执行一个特定的程序，并通过参数控制程序的执行过程。所以在开始操作系统调度之前，我们要先介绍Kettle的命令行执行方式。

通过命令行执行Kettle转换和作业

Kettle的Kitchen和Pan工具是Kettle的命令行执行程序。实际上，Pan和Kitchen只是在Kettle 执行引擎上的包装。它们只是解释命令行参数，调用并把这些参数传递给Kettle引擎。

Kitchen和Pan都通过shell脚本的方式启动，在Windows系统下，脚本名称是Kitchen.bat和Pan.bat，在类UNIX系统下，脚本名称是Kitchen.sh和Pan.sh。在执行这些脚本以及Kettle带的其他脚本时，要把Kettle目录切换为控制台的当前目录。

有时候，你可能要自己修改Kitchen或Pan脚本。在下面几种情况下，你可能要自己修改脚本：转换里如果使用了一些自定义的插件、自定义的Java 表达式，如果依赖其他的class，就需要相应修改 Java 启动时的classpath。如果发生了内存溢出的错误，可能要调整Java 启动命令行，设置新的Java虚拟机内存大小。第3章的“Kettle Shell脚本”解释了Kettle shell脚本的结构和内容。

说明：类UNIX系统的脚本默认情况下，是不能执行的，必须使用chmod 命令使脚本可执行。

命令行参数

Kitchen和Pan的命令行包含了很多参数，在不使用任何参数的情况下，直接运行Kitchen和Pan 会列出所有参数的帮助信息。参数的语法规则如下：

[/-]name [[:=]value]

参数以斜线 (/) 或横线 (-) 开头，后面跟参数名。大部分参数名后面都要有参数值。参数名和参数值之间可以是冒号 (:) 或等号 (=)，参数值里如果包含有空格，参数值必须用单引号 (') 或双引号 (") 引起来。

说明：使用横线和等号来设置参数，在Windows系统下会引起一些问题。在使用参数时最好都用斜线和冒号。

作业和转换的命令行参数非常相似，这两个命令的参数可以分为下面几类：

- 指定作业或转换
- 控制日志
- 指定资源库
- 列出可用资源库和资源库内容

表12-1列出了Pan和Kitchen共有的命令行参数。

表12-1 Pan和Kitchen共有的命令行参数

| 参数名 | 参数值 | 作用 |
|-------|--------|-------------|
| norep | | |
| rep | 资源库名称 | 要连接的资源库的名称 |
| user | 资源库用户名 | 要连接的资源库的用户名 |

续表

| 参数名 | 参数值 | 作用 |
|---------|---|-----------------------|
| pass | 资源库用户密码 | 要连接的资源库的用户密码 |
| listrep | | 显示所有的可用资源库 |
| dir | 资源库里的路径 | 指定资源库路径 |
| listdir | | 列出资源库的所有路径 |
| file | 文件名 | 指定作业或转换所在的文件名 |
| level | Error Nothing Basic Detailed Debug Rowlevel | 指定日志级别 |
| logfile | 日志文件名 | 指定要写入的日志文件名 |
| version | | 显示Kettle的版本号、build 日期 |

尽管Kitchen和Pan命令的参数名基本相同。但这两个命令里的dir参数和listdir参数的含义有一些区别。对Kitchen而言，dir和listdir参数列出的是作业的路径，Pan命令里的这两个参数列出的是转换路径。

使用Kitchen运行作业

除了共有的命令行参数外，Kitchen 还有几个自己特有的命令行参数，见表12-2。

表12-2 Kitchen 独有的命令行参数

| 参数名 | 参数值 | 作用 |
|----------|-----|--------------|
| jobs | 作业名 | 指定资源库里的一个作业名 |
| listjobs | | 列出资源库里的所有作业 |

下面的代码，是Kitchen 命令行的一个例子。

```
#
# list all available parameters
#
Kettle-home> ./kitchen.sh

#
# run the job stored in
# /home/foo/daily_load.kjb
#

Kettle-home> ./kitchen.sh \
    > /file:/home/foo/daily_load.kjb

#
# run the daily_load job from the
# repository named pdirepo
#

Kettle-home> ./kitchen.sh /rep:pdirepo \
    > /user:admin \
    > /pass:admin \
    > /dir:/ /job:daily_load.kjb \
```


使用Pan运行转换

除了和Kitchen共有的命令行参数外，Pan特有的命令行参数，见表12-3。

表12-3 Pan特有的命令行参数

| 参数名 | 参数值 | 作用 |
|-----------|-----|--------------|
| trans | 转换名 | 指定资源库里的一个转换名 |
| listtrans | | 列出资源库里的所有转换 |

用户自定义命令行参数

当执行转换或作业时，使用自定义的命令行参数给作业或转换传递参数会更方便。可以使用类似Java虚拟机参数的设置方式来设置用户定义参数。语法如下：

```
-D<name>=<value>
```

下面的例子演示如何在 Kitchen 命令行中设置用户自定义参数：

```
Kettle-home> kitchen.sh /file: -Dlanguage=en
```

在转换里，可以使用“获取系统信息”步骤来获取命令行参数。这个步骤在输入类别下。这个步骤可以获取到一些系统信息，并把获取到的信息保存到字段里，如图12-1所示。



图12-1 使用“获取系统信息”步骤获得命令行参数

在系统信息类型的列表里，可以看到有“命令行参数1”~“命令行参数10”的命令行信息。把字段设置为这种命令行参数，字段就可以读取相应的命令行参数。

类UNIX系统的调度命令：cron

在类UNIX系统里，cron是经常使用的调度命令，是系统自带的。

crontab是cron的调度文件，只要往这个文件里增加一个记录项，就会增加一个调度。crontab文件通常位于/etc/crontab，不同UNIX系统下crontab文件位置会有一些差别。

crontab 里的记录项就是一个字符串，这个字符串的前半部分是调度方式，后半部分是要调度的命令行，也就是Kettle的kitchen（执行作业）和pan（执行转换）命令，或者是调用kitchen或pan的shell脚本。

前半部分的调度方式字符串里包括五个日期时间值的字段，使用空格分隔。从左到右这五

个字段如下。

- 分钟: 0 ~ 59
- 小时: 0 ~ 23
- 日期: 1 ~ 31
- 月份: 1 ~ 12
- 星期几: 0 ~ 6, 0代表星期日, 1代表星期一, 等等。

例如下面的例子:

```
0      1      ?      *      5      run_kettle_weekly_invoice.sh
```

在这个例子里, 从左到右, 0代表0分钟, 1代表上午1点钟, ?代表一个月的任意一天, *代表每个月, 5 代表星期五。用语言描述就是每周五的上午1点钟。

说明: 关于cron和crontab的更多选项, 请使用 `man crontab`参考操作系统的帮助。网上也有很多相关的资源。

Windows系统下的at命令和计划任务

Windows系统下的调度一般使用at命令或计划任务。下面是at命令的例子:

```
at 00:00 /every:M,T,W,Th,F,S,Su "D:\pentaho\pdi\dauly_job.bat"
```

为了避免直接在at命令里使用一个长的pan或kitchen的命令行, 一般是写一个bat文件, 让at命令去执行这个bat文件 (在类UNIX系统下也一样, 写一个bash或sh命令)。

Windows 也提供了调度的图形界面, 在控制面板里有计划任务。

说明: 关于at 命令或计划任务的更多使用方法, 请访问 <http://support.microsoft.com> 搜索 “at 命令” 或 “计划任务”。

12.1.2 使用Pentaho 内置的调度程序

如果你使用了Pentaho BI 套件, 可以用它内置的调度程序来带调度Kettle的转换或作业。

说明: 关于Pentaho BI服务器的使用和开发不在本书讨论的范围内。感兴趣的读者可以参考由 Roland Bouman和Jos van Dongen 编写的*Pentaho Solutions*一书。

Pentaho BI服务器使用了Quartz的企业调度器, Quartz是Open Symphony 项目的一部分。关于Quartz和Open Symphony参考<http://www.opensymphony.com/quartz/>。(在翻译本书时, Open Symphony已经关闭, 其下的各子项目, 分散在不同的组织或公司。Quartz被商业公司收购。——译者注)

使用Pentaho 自带的调度器运行Kettle 作业或转换需要先做两件事:

- 创建动作序列 (action sequence), 动作序列是一个任务容器, 由BI服务器解释执行。
- 创建一个调度, 调度决定何时由Pentaho BI服务器执行一个动作序列, 以及执行的频率。

做好上述两件事后, 就可以把调度和动作序列绑定, 动作序列就可以按照调度来执行。

创建运行Kettle 转换和作业的动作序列

由Pentaho BI Server 执行Kettle作业调度的第一件事情就是要创建一个动作序列，动作序列里可以是Kettle 资源库形式的作业或文件形式的作业。

如果使用Kettle 资源库形式的作业，就要编辑Pentaho BI Server的pentaho-solutions/system/kettle 目录下的settings.xml文件。这个文件的内容如下：

```
<kettle-repository>

  <!-- The values within <properties> are
        passed directly to the
        Kettle Pentaho components. -->

  <!-- This is the location of the
        Kettle repositories.xml file,
        leave empty if the default is used:
        $HOME/.kettle/repositories.xml -->

  <repositories.xml.file></repositories.xml.file>
  <repository.type>files</repository.type>
  <!-- The name of the repository to use -->
  <repository.name></repository.name>

  <!-- The name of the repository user -->
  <repository.userid>admin</repository.userid>

  <!-- The password -->
  <repository.password>admin</repository.password>

</kettle-repository>
```

要在这个配置文件的<repository.name>节点内设置资源库的名称，注意资源库的名称要与repositories.xml文件里定义的资源库的名称一致。关于repositories.xml文件，请参考第3章。也可以不用默认的repositories.xml文件，此时就要设置 <repositories.xml.file>节点。另外资源库的用户名和密码节点<repository.user>和<repository.password>也要正确设置。

在动作序列 (action sequence) 里定义Kettle转换

Pentaho Design Studio是action sequence文件的配置工具，在添加动作的配置窗口里有很多动作类型。在Get Data From 类型下，可以找到Pentaho Data Integration这一动作。

Pentaho Data Integration动作可以从动作序列里接收参数，并返回一个结果集，以便动作序列后续使用。除了结果集，动作序列还能获得一些诊断信息，如转换的日志。

Pentaho BI Server 提供了一些演示例子。在Pentaho BI Server的bi-developers\etl 目录下找到一些例子。其中PDI_Inputs.xaction 就是其中的一个例子，如图12-2所示。

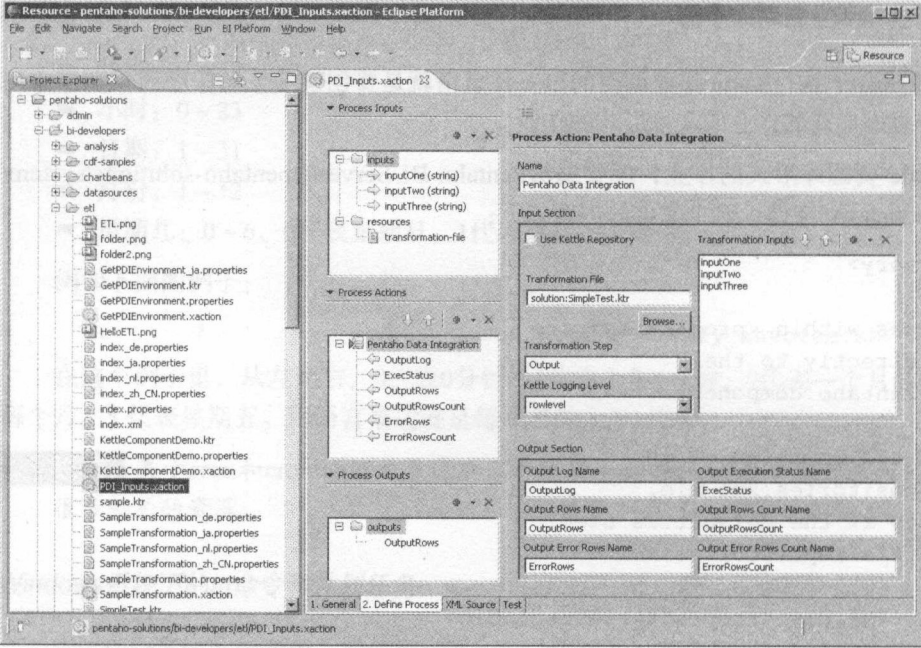


图12-2 PDI_Inputs.xaction的配置窗口

下面是配置 Pentaho Data Integration动作的一些选项：

- 如果要运行Kettle资源库里的转换，选中“Use Kettle Repository”选项。当选中这个选项后，就可以输入资源库里的路径名和资源库里的转换名。
- 在转换文件输入框里，可以输入转换文件名。使用“Browse”按钮查找BI Server所在机器上的.ktr文件。如果选择的是本地的任何一个文件，转换文件名是以 file:// 开头的URL。也可以选择Pentaho 资源库里的一个文件，此时就要使用前缀solution:。图12-2例子里的SimpleTest.ktr文件就是在Pentaho资源库和动作序列文件 PDI_Inputs.xaction 在同级目录下的文件。
- 在转换步骤输入框里，输入你想获取结果集的步骤名称，从这个步骤输出的所有记录将返回给动作序列。尽管转换可以产生很多个结果集，但在这里只能指定一个。
- 转换输入列表用来给转换提供参数。Kettle可以通过“获取系统信息”步骤获取到这些命令行参数。在本章的图12-1 里曾讲过这个步骤。
- 在 Kettle 日志级别列表里可以设置执行转换后产生的日志的级别。日志级别在本章后面讲述。
- 在窗口的最下面，可以映射一些统计字段，如执行状态、日志、返回的记录行数等。

使用管理控制台创建和维护调度

Pentaho BI Server里带有管理控制台，通过管理控制台可以创建和维护调度。启动BI Server后，打开管理控制台页面，单击调度标签。可以看到一组公有和私有的调度，如图12-3所示。

默认情况下，有一个PentahoSystemVersionCheck调度。这个调度是一个私有的调度。用来定期检查Pentaho是否有新版本。

说明： 共有调度是所有用户都可以看到和使用的调度。私有调度不能被所有用户看到，一般是系统维护调度，只能被管理员看到。

在调度标签里，单击工具条里的第一个按钮可以创建一个新的调度。创建调度对话框如图12-4所示。

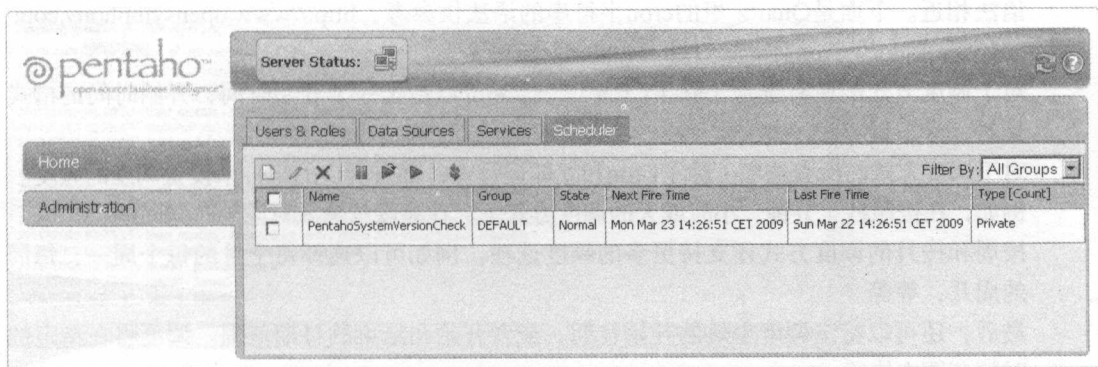


图12-3 Pentaho管理控制台的调度标签

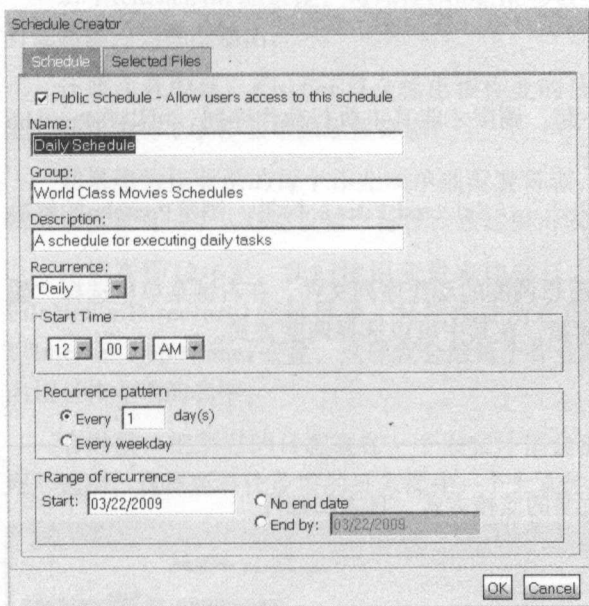


图12-4 创建调度对话框

创建调度对话框里的选项如下。

- 公有调度复选框：用来创建一个公有调度。
- 调度名称输入框：注意所有用户都可以看到公有调度的名称，名称要尽量简洁和清晰。
- 分组：为调度分组，可以根据部门分组（例如，仓库、人力资源），根据地理位置分组（艾德蒙顿、洛杉矶），根据主题分组（销售、市场）或根据时间分组（每天、每周）。

调度标签的工具条上有一个列表，这个列表可以根据分组过滤调度，便于管理人员管理。

说明：即使你不想把这些调度分组，你也需要输入一个分组名。下面讲的描述字段（描述字段）也如此。

- 描述字段：描述字段里可以输入一些调度的描述信息，一般是说明调度的目的。
- 调度方式下拉列表：调度方式下拉列表有各种调度方式，从按秒调度到按年调度。另外

也有立即执行选项。如果这些选项还不能满足调度的需求,你还可以选择Cron选项,然后定义一个cron字符串。这里的cron字符串的语法和标准的UNIX系统的cron命令的语法相近。下面是Quartz里的cron字符串的语法供参考, <http://www.opensymphony.com/quartz/wikidocs/CronTriggersTutorial.html>。

对于调度方式的所有选项(除了Cron),你都可以指定一个开始时间,开始时间的格式是时、分、秒。

对于调度方式的所有选项(除了Cron和立即运行),都有对应的组件来设置调度的频度。

例如,在按照秒、分钟、小时或天的调度方式下,在频度里输入对应的时间间隔。

按周和按月的调度方式还支持更多的频度选项。例如可以选择每个月的每个周一、每周的周几,等等。

最后,还可以指定调度生效的开始日期,或者开始和结束的日期范围。调度将在指定的时间范围内执行。

设置完调度后,单击OK按钮保存设置并关闭对话框。

将动作序列和调度方式绑定

只有把新创建的调度和动作序列绑定到一起,调度才能真正执行动作序列。可以在Pentaho管理平台或Pentaho用户控制台来完成该操作:

- 在图12-4的对话框里,选择“选中的文件”(Selected Files)标签。浏览Pentaho的解决方案资源库,选择要调度的动作序列。
- 在Pentaho的用户控制台,鼠标右键单击想调度的动作序列文件。在右键菜单里选择“属性”。在属性对话框里选择“调度”标签,在其中可以选择调度方式。

12.2 监控

Kettle作业通过调度方式运行后,有两种主要的监控方式,日志和邮件。

12.2.1 日志

Kettle的日志框架在转换或作业执行时输出运行过程的反馈信息。日志对于监控程序和调试是非常有用的。

日志将在第14章中详细讨论,在本章将只介绍日志的一些基本知识。

查看日志

在Spoon里,在工作区窗口下面的执行窗口的日志标签下可以看到日志。日志标签如图12-5所示。

日志标签实际就是一个列表框,日志显示在这个列表框里,另外还有几个按钮。错误的日志用红色表示。如果只想显示错误的日志,可以单击工具条上的那个红色按钮。在红色按钮旁边的是清除按钮,最后是一个设置按钮,用来设置日志的显示方式,如设置日志级别、设置每行日志前是否有日期和时间等。

在开发和测试过程中，执行窗口里的日志非常有用。而在部署和调度环境里，没有什么用处，因为在生产环境下，都是通过命令行执行作业。

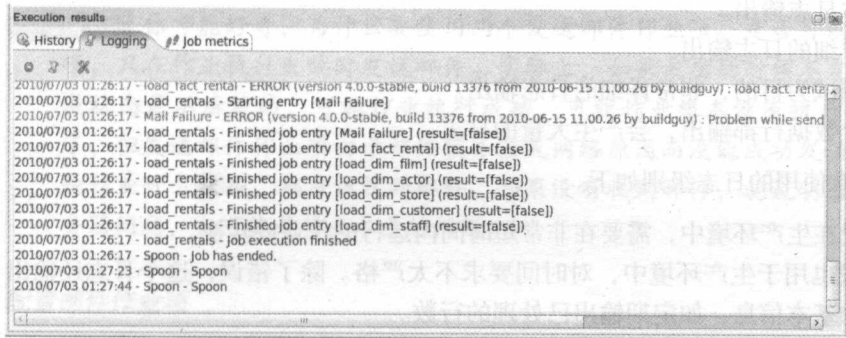


图12-5 在 Spoon 窗口里查看日志

对于Kitchen和Pan来说，可以使用logfile参数来指定一个日志文件。如果不指定日志文件，日志将输出到标准输出。

还要注意的，Kitchen只会输出根作业的日志。如果想输出根作业下的每个子作业的日志，需要为每个子作业单独配置日志。

如果想为作业里的每个作业项单独配置日志，右键单击作业项选择“编辑作业项”，在编辑对话框里有一个“日志设置”标签，在这里可以指定日志文件的位置，如图12-6所示。

注意在图12-6里，我们使用变量来构造日志文件名。在这个例子里，在`${Internal.Job.FileName.Directory}`变量目录（外层作业所在的目录）下创建文件。对于文件名，我们使用`${Internal.Step.Name}`变量，这样就会给每个作业项创建一个独立的日志文件，每个日志的文件名就是作业项的名字。

有时候，把所有的日志放在一个大文件里会更方便，这时我们需要选中“添加到日志文件尾”选项。这样所有作业项的日志都在一个大文件里。

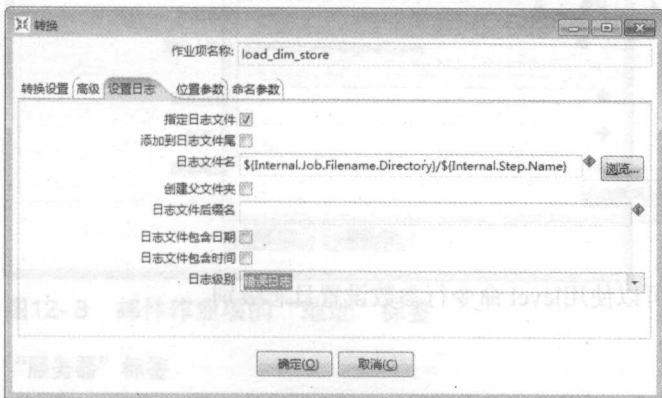


图12-6 配置每个作业项的日志位置

日志级别

Kettle 支持很多种日志级别来控制日志的复杂度。下面的日志级别依次增高。

- 没有日志：没有任何输出。

- 错误日志: 指输出错误日志。
- 最少日志: 最少日志输出。
- 基本日志: 基本日志输出。
- 详细日志: 更详细的日志输出。
- 调试日志: 以调试为目的, 非常详细的日志输出。
- 行级日志: 每个数据行都输出, 会产生大量的日志。

在实际中, 我们最常使用的日志级别如下。

- 错误日志: 一般在生产环境中, 需要在非常短时间内运行的作业或转换。
- 基本日志: 一般也用于生产环境中, 对时间要求不太严格。除了错误, 这个日志还输出程序运行状态的基本信息, 如定期输出已处理的行数。
- 行级日志: 最详细的日志, 一般只用于开发和测试阶段。

我们都知道, 日志会消耗系统的性能: 日志输出越多, 性能下降越严重。所以不能只为了安全而输出最详细的日志, 需要根据手中的任务选择不同的日志级别。

在转换和作业的执行窗口里, 我们也可以设置日志级别, 一个设置了日志级别的执行窗口如图12-7所示。

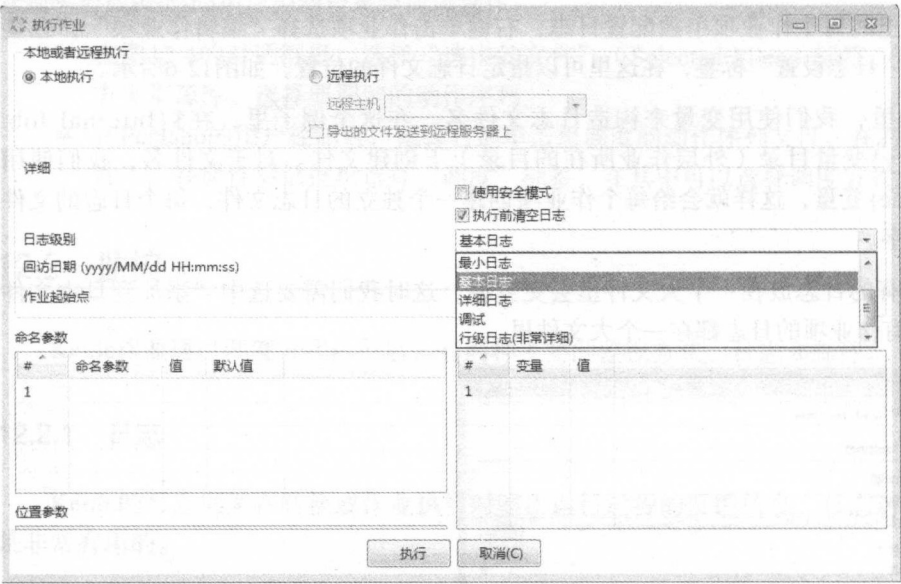


图12-7 设置了日志级别的执行窗口

对Kitchen和Pan 命令行工具而言, 可以使用level 命令行参数设置日志级别。

用户自定义的日志

可以使用转换或作业里的“写日志”步骤或作业项, 把你自定义的日志写到日志文件里。

12.2.2 邮件通知

使用作业里的“发送邮件”作业项通知管理员作业是否完成、是否有错误发生, 也是一个简单的方法。在第4章的图4-6里的load_rentals.kjb 就是一个“发送邮件”作业项的简单例子。

load_rentals.kbj作业里有两个发送邮件作业项。“Mail Success”作业项是在load_fact_rental转换执行成功后发送邮件，“Mail Failure”作业项是在作业里某个转换失败后发送邮件。

说明：你可能好奇，为什么要使用两个发送邮件作业项。是否可以只用一个发送邮件作业项，只在作业执行失败时发送邮件。实际上，如果成功的时候不发送邮件，那么作业的完成状态是不清楚的，可能作业执行成功，可能作业根本没有执行，也可能作业执行失败了，但发送邮件作业项由于邮件服务器或网络原因而没能成功发送邮件。我们必须确保无论作业什么状态，都应该收到邮件。如果没有收到邮件，就说明出问题了，可能是邮件服务器问题，也可能是作业本身的问题。

配置邮件作业项

尽管邮件作业项的配置项多一些，但配置并不麻烦，主要的配置项如下。

“地址”标签

在“收件人地址”输入框里必须指定至少一个有效的E-mail地址。抄送和暗送输入框不是必须输入的项。“发件人”栏里的“发件人地址”和“回复名称”都是必须输入的项。下面的“回复地址”、“联系人”等选项都是可选项。在一些常见的成功/失败发送场景下，收件人应该是IT支持人员，指定数据集成团队的某个人作为发送人。图12-8显示了对话框的“地址”标签。

图12-8 邮件作业项的“地址”标签

“服务器”标签

必须指定SMTP服务器的配置信息，如图12-9所示。

至少要设置SMTP服务器的地址。端口是可选项，默认是25（SMTP的默认端口号）。在大多数情况SMTP服务器都需要验证，要选中“用户验证”选项，并输入用户名和密码。如果SMTP服务器使用了安全验证协议，如SSL（Secure Sockets Layer）或TLS（Transport Layer Security），你还要选中“使用安全验证”复选框，并选择正确的安全连接类型。注意如果使用了安全验证协议，SMTP服务将使用其他的端口号。如安全验证协议是SSL，则对应的端口号是465。

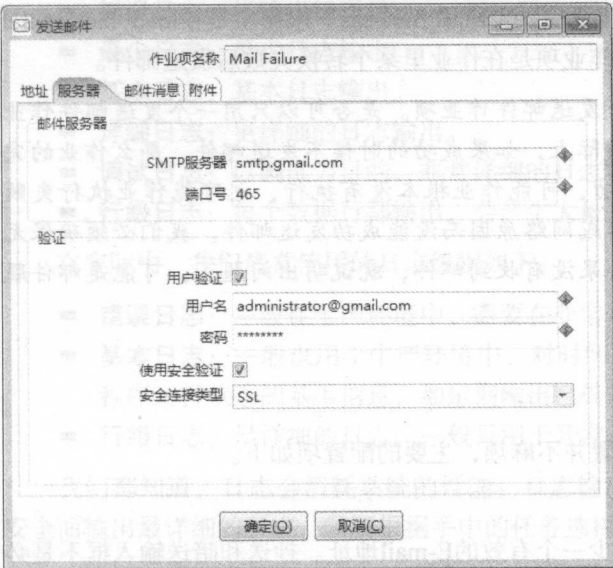


图12-9 邮件作业项的“服务器”标签

“邮件消息”标签

邮件的正文要在“邮件消息”标签中设置，如图12-10所示。

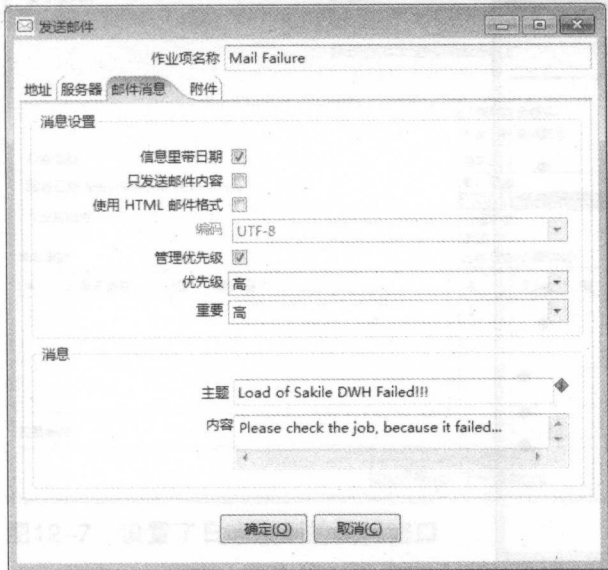


图12-10 邮件作业项的“邮件消息”标签

在“主题”和“内容”输入框里分别输入邮件的标题和正文。设置时可以使用文字或变量。默认情况下，Kettle会在邮件正文的末尾追加一个简要的作业运行状态报告。如果不要这个状态报告，选中“只发送邮件内容”选项。另外，你还可以选中“使用HTML 邮件格式”来发送HTML格式的邮件。一些邮件客户端可以读取邮件报头头的优先级参数，所以你还可以设置“管理优先级”属性，使发送的邮件带有不同的优先级。

“附件”标签

在附件标签里可以设置哪些文件可以作为邮件的附件，附件标签的设置如图12-11所示。

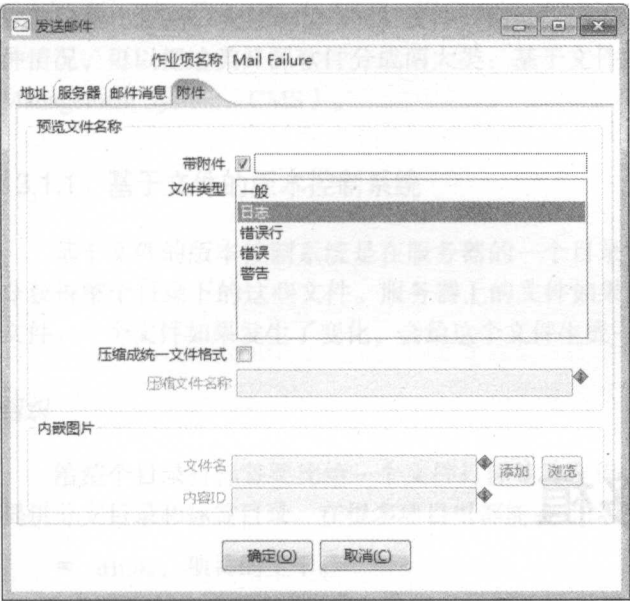


图12-11 邮件作业项的“附件”标签

如果要让邮件带附件，首先要选中“带附件”复选框，同时在“文件类型”列表框里选中一个或多个要作为附件的文件类型。注意如果选中了“日志”文件类型，需要事先为这个作业项设置日志（参考图12-6）。

选中“压缩成统一文件格式”，可以把所有的附件压缩成一个压缩文件。如果选择了压缩，必须要在“压缩文件名称”里设置一个文件名。

12.3 小结

在本章，我们讲述了ETL在生产环境中的两个主要内容，调度和监控。我们主要学习了：

- 运行作业和转换的Kitchen和Pan命令行。
- 操作系统调度，例如cron、at和Windows计划任务。
- 使用Pentaho BI Server里的Quartz 调度。
- 控制Spoon、Kitchen和Pan 里的日志文件和日志详细级别。
- 使用电子邮件作业项做作业监控。
- 如何把Kettle 日志加入到邮件通知中。

第13章 版本和移植

前面我们讲过了ETL设计、构建和部署，并讲解了如何在单用户开发环境下使用 Kettle。在现实情况中，一个项目里有多个ETL开发人员，这就需要使用版本控制系统来管理每个人提交的ETL作业版本。另外在大多数项目里还要把开发、测试、发布、生产环境区分开。下面的ETL子系统涵盖了这些需求。

- 子系统 25：版本控制系统。
- 子系统 26：从开发环境到测试环境再到生产环境的版本移植系统。

在本章，我们讨论一下为什么要使用版本控制系统，并介绍几个开源的版本控制系统。然后我们讲述 Kettle 的元数据，以及 Kettle 元数据可以用哪些形式来表现，最后说明如何利用 Kettle 的元数据来进行版本控制和移植工作。

13.1 版本控制系统

如果只是一个人，或一个很小的团队来开发一个ETL系统，ETL过程是易于维护的，容易确定哪个开发人员做了什么。但若项目规模变大，参与的开发人员变多，事情就完全不同了。为了能让ETL在大规模的项目里也易于维护，对ETL过程的任何小的修改都应该被跟踪到。

如果你的ETL系统在运行过程中发生了问题，你通常被问到的第一个问题是“你最近改什么了？”你的回答往往是“我什么也没干。”其实在一个系统里，变化包括多种形式。如果能知道问题的起源，修改问题相对容易一些，否则我们只能去debug了。

为避免由于某些变化而导致的问题，我们需要变更记录系统。这样的系统就是版本控制系统（Version Control System，VCS）或修订控制系统（Revision Control System）。在过去，VCS

软件价格比较高，大公司里 VCS 软件只提供给ETL的开发团队。但开源软件的出现打破了这种情况，可以把这类开源软件分成两大类：基于文件的版本控制系统和内容管理系统（Content Management System, CMS）。

13.1.1 基于文件的版本控制系统

基于文件的版本控制系统是在服务器的一个目录结构下组织管理文件的。可以通过检出命令获得整个目录下的这些文件。服务器上的文件如果发生了变化，可以通过更新命令更新相应文件。一个文件如果发生了变化，会给这个文件生成一个新的版本号。

组织

给整个目录打标签要比给一个文件打标签重要得多，所以除了主干目录，VCS 软件通常都提供分支目录和标签目录。在很多项目里都能看到这种树状结构。

- trunk/：项目的主干。
- branches/：项目的分支，分支一般属于项目里的某个独立版本，里面包含该版本相应的文件。
- tags/：标签是项目文件的快照。它只是一个标签，便于管理。

如果你看了Kettle的源代码树（<http://source.pentaho.org/svnkettleroot.Kettle>），可以看到文件就是以这种方式保存的。在编写本书的时候，trunk/分支包括了Kettle的最新开发版本（4.1.0的Milestone 版本）。而branches/目录下则保存了从Kettle 2.2.2 到 4.0.0的所有版本。tags目录下的所有目录是一些临时快照，便于开发、测试。

主流的基于文件的VCS系统

下面列出了一些主流的基于文件的VCS系统。

- CVS：第一个流行的版本控制系统是CVS，CVS采用的是GNU的公共协议（GPL）。起源于25年前的一些脚本。在很长一段时间里，CVS 都是项目里默认的版本控制系统。它可以让一个大的开发团队有效工作。
CVS的主要缺陷就是缺少原子性，缺少Unicode支持，缺少对二进制文件的支持和烦琐的分支和标签操作。而且改了文件名以后文件的历史都会丢失。特别是缺少原子性在一些情况下会导致CVS 资源库的崩溃。在资源库上工作的用户越多，这种可能性就越大。尽管存在这些问题，使用CVS 强于不使用任何VCS系统。因为CVS的知名度很高，几乎各种操作系统下的CVS 版本还在被维护。在SourceForge.net 等一些开源下载平台上，CVS还提供下载。
- Subversion：CVS的缺陷使一批新的VCS系统得以出现和发展。当前最著名的就是Apache的Subversion。这个项目起源于2000年，项目的目标是创建一个最兼容的VCS版本，而没有CVS的那些缺陷。Subversion的开发人员基本完成了这个目标，Subversion开始流行。在编写本书时，Subversion可能是当前最流行的VCS系统。
Subversion的主要缺陷是对文件的重命名操作。和其他基于文件的版本控制系统一样，Subversion使用文件名处理基本的VCS操作。Kettle 项目也使用 Subversion来管理代码。

Subversion使用方便, 我们也推荐你在数据整合的项目中使用Subversion作为VCS系统。在编写本书时, Kettle还没有在设计器里整合Subversion功能, 但Kettle的作业和转换都可以保存成文件形式, 可以在Spoon设计完后, 再通过Subversion来管理这些XML文件。

说明: 当使用Kettle的文件资源库时, 默认选择就是保存成XML格式文件, 而在使用数据库资源库时, 可以通过选择文件→导出→到XML, 导出成XML格式的文件。

- 分布式版本控制系统: 这两年出现了所谓的分布式版本控制系统 (distributed version control system, DVCS)。这种系统和传统的基于客户端服务器的系统如CVS或Subversion的主要区别是DVCS可以有多个资源库。项目管理人员可以利用工具把不同资源库的代码合并到一起。实现了DVCS的开源系统有 Git、Mercurial、Bazaar和Fossil。使用Git的大的项目有Linux Kernel和Android; 使用 Mercurial的有Mozilla和OpenOffice.org, 使用Bazaar的有Ubuntu和MySQL。这些项目的规模都很大。

13.1.2 内容管理系统

内容管理系统 (CMS) 都是面向服务的系统。如 Documentum、Sharepoint、Alfresco、Magnolia、Joomla和Drupal。当然文件内容都可以从服务器下载, 内容管理系统的目的是使人们可以在一起工作。很多业务处理系统 (Business Management System, BPM) 和工作流系统 (Workflow Management System, WMS) 都在底层使用了CMS来管理内容。很多CMS系统都在内部使用了标识符而不是文件名来做操作。所以文件重命名、目录重构都非常容易。也可以为每个文件添加额外的信息, 如描述、图标和文件类型信息, 等等。这些特性都非常适合于管理Kettle元数据。

13.2 Kettle 元数据

在讲述Kettle的版本控制和移植等选项之前, 有必要阐述一下Kettle的元数据。

在本书前面曾经讲过, Kettle的转换和作业都是使用ETL元数据来描述的。让我们看看什么是ETL元数据, 元数据是一个宽泛的概念, 一般是指“描述性数据”或“数据的数据”, 对ETL元数据而言, 就是用来描述ETL要执行的任务。例如在Kettle里我们要描述下面的一些信息:

- 输入文件的格式。
- 要使用的关系型数据库的用户名。
- Web服务的URL。
- 要存储数据的关系表的名称。
- 读、转换、写数据的各种操作。
- 各种操作的执行顺序。

Kettle 里的执行引擎将会解释这些元数据, 并执行这些元数据定义的任务。

在Kettle 里ETL的元数据有多种形式, 当你进入Spoon就能看到元数据最直观的形式, 图形方式, 这种方式便于开发人员操作。元数据也可以在Kettle软件里以Java对象的方式存储 (第22章), 在设计完后, 转换和作业也能以XML格式文件的形式存储, 或存储到数据库中。

13.2.1 Kettle XML 元数据

XML是一种支持Unicode的数据接口，XML也成为Kettle元数据的一种表现形式。XML不便于人阅读，只便于开发使用，但在一些情况下开发人员也可以手工修改XML文件。幸运的是，Spoon可以把Kettle的各种元数据都保存为XML格式，所以设计人员可以不用和XML格式的元数据打交道。作业和转换的XML格式的元数据其实也比较简单，在XML元数据里只使用了XML节点，而没有使用节点的属性。

Kettle里两种最重要的元数据格式是作业和转换，它们都可以保存成XML格式的文件。下面先介绍转换的XML格式。

转换XML

每个XML文件都要有一个根节点，.ktr转换文件的XML的根节点必须是<transformation>。如果不是，这个文件就不是Kettle转换文件。根节点下面一层的节点如下（下面节点省略了左右尖括号）。

- Info：包含了转换的信息，例如转换名、描述等。
- trans-log-table：包含转换日志表的设置。
- perf-log-table：包含性能日志表的设置。
- channel-log-table：包含通道日志表的设置。
- step-log-table：包含步骤日志表的设置。
- notepads：包含转换里的所有注释。
- connection, slaveservers, clusterschema, partitionschema：依次描述了数据库连接、子服务器、集群模式、分区模式。这些节点都可以有多个，如一个转换实例有多个数据库连接或集群服务器。
- order：描述了步骤之间连线的方向。
- hop：描述了步骤之间的连线信息，hop节点是order节点的子节点。
- step：包含了指定步骤的元数据，可以有多个步骤节点。根据步骤的不同，step节点的结构也不同。因为有的步骤是以插件方式提供的，在插件里就要定义该步骤的XML结构。
- step_error_handling：这个节点包含了error子节点，error子节点里指定了错误处理的源和目标步骤，该节点里还其他错误处理的信息，如最大允许错误数等。

上面很多节点都是可选的，例如，若没有定义转换日志表，trans-log-table节点就可以没有。

作业XML

作业XML的根节点是<job>。作业文件的结构和转换的结构稍有不同，但很多子节点都是类似的。

- name, description：和转换文件不同的是，作业文件里没有info节点，而是直接列出了名称、描述等节点。
- job-log-table：包含作业日志表设置。
- channel-log-table：包含通道日志表设置。
- jobentry-log-table：包含作业项目日志表设置。

- notepads: 包含作业里的所有注释。
- connection和slaveservers: 包含数据库连接和子服务器信息, 可以有多个。
- hops: 描述了作业项之间的连线信息。
- entries: 包含了指定作业项的元数据, 作业项也可以通过插件扩展, 作业项不同, entries节点的结构也不同。相同作业项的entries节点的结构相同。

对一般的Kettle 用户来说, 不必了解Kettle XML 元数据的结构。但在一些场合下, 如果想手工生成或修改Kettle XML文件, 下面的例子可以给你提供一些帮助。

全局替换

如果你有几百个Kettle 转换或作业都保存成扩展名是.ktr或.kjb的XML格式的文件, 如果你要通过Spoon打开这些文件, 并修改其中的一个参数, 将会是很烦琐的事情, 而且容易出错。例如, 在开发完成后, 你发现生产环境的数据库使用的模式名和开发环境不同, 而在ETL流程里模式名dwh是硬编码的, 没有使用变量。这时就要写一个UNIX、Linux、OS X, 甚至是Windows (使用Cygwin) 环境下的shell脚本, 使用这个 shell脚本来替换转换里<schema>节点的内容。

```
# Prevent loss of information : stop after an error!
#
set -e
# Loop over all the transformation XML files:
#
for file in *.ktr
do
    # File name can contain spaces: quote them!
    #
    <"$file" \
        Sed 's/<schema>dwh</schema>/<schema>${DWH_SCHEMA}</schema>/g' \
    > TEMPFILE

    rm "$file"
    mv TEMPFILE "$file"
done
```

这个脚本名为globalreplace.sh, 可以在本书的下载文件中找到, 这个脚本可以将所有转换文件里的<schema>dwh</schema> 替换为 <schema>\${DWH_SCHEMA}</schema>, 也就是在转换里的所有模式名都使用变量DWH_SCHEMA。对于熟悉Shell(bash)、awk、Perl、Ruby等脚本语言的开发人员来说, 可以使用脚本语言快速解决这些小问题。这段脚本里最有用的命令是sed, 也就是stream editor的缩写。sed是.nix系统下处理文本的一个常用命令。关于sed 命令参考 <http://www.grymoire.com/unix/Sed.html>。

13.2.2 Kettle 资源库元数据

从Kettle 4 版本以后, 资源库类型也做成了插件。也就是说Kettle元数据可以是任何可能的格式。Repository 接口里包括了所有必要的方法, 来串行化转换、作业、以及数据连接、子服务器等各种对象。为了便于ETL开发人员给这些对象分类, Kettle 资源库也按照目录划分, 下面我们

看看Kettle 里常用的几种资源库。

数据库资源库

Kettle的数据库资源库就是把Kettle的元数据串行化到数据库中，使用数据库表来保存转换或作业里各个组件的配置信息。例如，在数据库资源库里的R_TRANSFORMATION 表保存了Kettle 转换的名称、描述等属性。R_STEP 表保存了转换里的所有步骤。每个表都有标识列，可以把Kettle里的所有对象关联起来。

通过数据库资源库，ETL开发人员可以在一起开发。数据库资源库成为一个ETL元数据的存储中心。在Spoon可以创建新的资源库，也可以升级现有资源库。如图13-1所示，在“工具→资源库→连接资源库”里创建资源库。

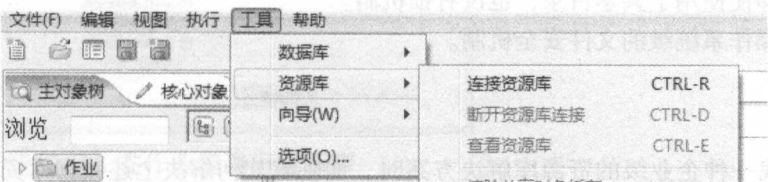


图13-1 连接到资源库

选择上面的菜单命令后，会出现“资源库连接”对话框，如图13-2所示，这个对话框在Spoon 启动时也可以出现。

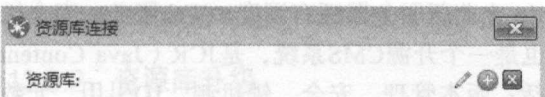


图13-2 管理资源库

单击 + 号按钮，列出一组可用的资源库类型，如果你选择的是数据库资源库，会出现下面的对话框。

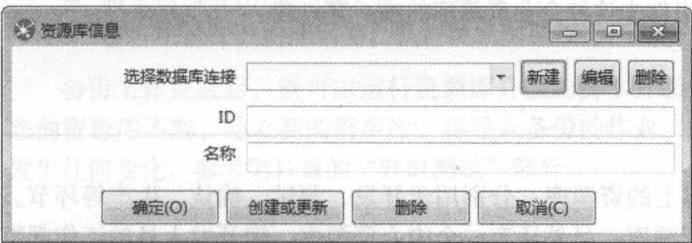


图13-3 Kettle数据库资源库对话框

在这个对话框里，要设置资源库所要使用的数据库连接。系统在选定的数据库连接里会自动创建需要的数据库表和索引。

尽管Kettle数据库资源库可以很好工作，但它还是有下面的一些缺点。

- 不能存储转换或作业的多个版本。
- 严重依赖于数据库的锁机制来防止工作丢失。
- 没有考虑到团队开发，开发人员不能锁住某个作业自己开发。
- 安全机制简单，依赖于数据库的安全机制。

Kettle文件资源库类型

Kettle文件资源库使用了Kettle的XML存储文件。它使用Apache VFS驱动（见第2章）来访问Kettle元数据。也就是说文件可以不在本地，可以在FTP、HTTP或者是.zip的压缩包。

创建文件资源库也很简单，在创建资源库时，选择Kettle文件资源库类型，并指定一个目录。

因为保存的是XML格式的文件，这种资源库也有XML相关的一些缺点。

- 对象（如转换、作业、数据库连接等对象）之间的关联关系难以处理，所以删除、重命名等操作会比较麻烦。
- 没有版本历史。
- 难以进行团队开发，即使使用了共享目录，也没有锁机制。
- 没有安全机制，只有操作系统级的文件安全机制。

Kettle 企业级资源库类型

Pentaho的开发团队在寻找一种企业级的资源库解决方案时，面临着如何解决上述数据库资源库和文件资源库缺点的问题。也要考虑到目前流行的各种VCS系统的缺点。尤其是文件重命名和文件移动的缺点，以及如何在VCS上实现安全层。

为了解决这些问题，Pentaho选择了内容管理系统（CMS）这个方向。Kettle 企业版服务器上的资源库使用了Apache Jackrabbit，另外Kettle 企业版服务器还有调度、Web服务、安全机制等。Jackrabbit（<http://jackrabbit.apache.org>）也是一个开源CMS系统，是JCR（Java Content Repository）规范的Java实现。它的主要功能包括：版本管理、安全、锁机制、ID引用、元数据、查询、重命名、文档目录重构。Jackrabbit的这些功能通过Kettle数据整合服务器的一组Web服务（Web Services）对外提供，这样可以方便增加新特性，而不影响之前和服务器建立通信关系的客户端。Pentaho还计划让Pentaho的BI Server，或Pentaho的其他工具也支持这种资源库，但在编写本书的时候，Kettle是Pentaho组件中这种企业资源库的唯一客户端。

13.3 管理资源库

在大部分情况下，至少有两个以上的资源库：分别用于开发、测试、确认、生产等环节。通常，开发人员都会有自己的开发资源库，另外还有一个中心资源库，把开发人员的工作都组织起来。因为在项目生命期中，Kettle 资源库扮演了一个中心节点的作用，所以有必要定期通过“导出”备份资源库。也可以通过“导入”完成不同环境下的资源库的移植。

13.3.1 导出和导入资源库

Spoon的“工具→资源库导出”菜单命令的主要用途是备份资源库。备份资源库实际是把资源库串行化为XML文件。XML文件的根节点是<repository>，根节点下面有很多作业和转换子节点。

另外在作业里“导出资源库”作业项也可以导出资源库，如图13-4所示。

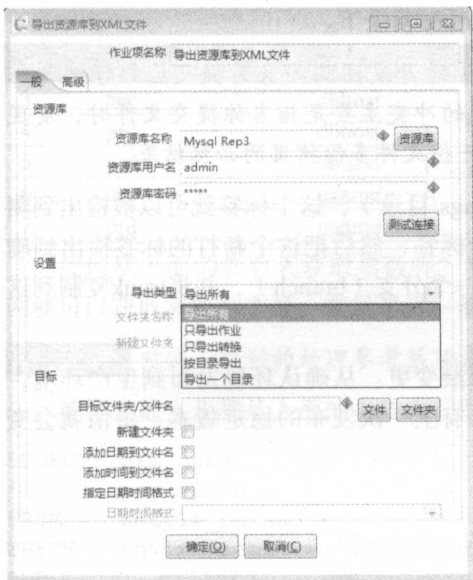


图13-4 在作业里导出资源库

通过这个作业项，可以定期自动导出资源库或资源库下的某个目录。

最后，你还可以使用 Pan 命令行工具导出资源库，命令行如下：

```
sh pan.sh -rep="Production" -user="admin" -password="admin"
-export="/tmp/export.xml"
```

13.3.2 资源库升级

如果在Kettle的新版本里，需要保存更多的元数据，你就要升级资源库。和其他软件升级一样，升级之前要做备份。备份时最好同时使用下面两种方法。

- 将资源库导出成XML文件（见前面）。
- 备份资源库数据库本身。执行关系型数据库的dump或export等命令。

备份工作完成后，就可以运行资源库升级的脚本升级资源库。还有一种方法，就是保留原来的资源库不变，建立新的资源库，再导入备份的作业，这种方法的优点就是原来的资源库不发生改变。参考第11章的“升级测试”部分。

13.4 版本移植系统

在软件项目的生命周期里，比较麻烦的一件事情就是将工作从开发环境移植到测试环境、确认环境以及最终的生产环境。Kettle移植面临的第一个问题就是：“如何把作业或转换从开发环境移植到生产环境？”这其实和元数据的保存方式有关。下面章节将说明处理XML文件和资源库的几个方法。

13.4.1 管理XML文件

如前面介绍的，管理转换或作业文件的最好方法是把它们放在版本控制系统（VCS）里。

这样可以在多台机器上做开发，并把开发的成果保存在trunk/目录下。

警告：如果你要频繁地修改已创建的作业，提醒你的开发团队要及时提交已经修改的作业，这样可以将冲突降低到最小程度。我们这里说的冲突主要是指当你提交文件时，发现这个文件已经被别人检出并提交过了。这种冲突是一份文件多份拷贝的必然结果。

当要测试时，可以把 trunk 分支打个标签（复制到tags/目录），这个标签就可以被检出到测试服务器。当你改完测试人员提出的问题后，再打一个标签，然后把这个新打的标签检出到确认服务器，做验证。最后，当你准备发布时，需要新建一个分支（branch），并将trunk复制到这个分支下，发布到生产环境。

上面的流程非常简单，使用VCS的另一个好处是回滚变更。从确认环境发布到生产环境，总是有风险的。如果发布后发生了一些意外情况，如果有上一次发布的稳定版本，事情就会变得容易些。

13.4.2 管理资源库

如果使用数据库资源库，要管理好开发、测试、确认、生产环境，最简单的方法就是为每个环境建立一个资源库。

警告：Kettle 社区版本的数据库资源库不提供锁和版本管理等功能，每个人都要建立自己的资源库，这样不适合团队开发。Kettle 企业版可以提供锁和版本管理等功能。

把变更从一个环境移植到另一个环境，可以有很多方法，最简单的是资源库的导出和导入。第二个方法是在Spoon 里打开一个转换或作业，然后断开当前资源库连接，再打开新的资源库连接，再保存。如果开发团队的人员较多，要把每个人资源库的作业或转换提交到测试资源库，最好指定一个人来统一提交到测试数据库，这样可以避免冲突。

13.4.3 解决方案参数化

如果要想移植的工作量最小，最好的方法是参数化。就是说对每个环境里不同的设置都要使用变量或参数。尤其要注意的是数据库连接参数、文件路径。文件路径要使用相对路径，例如内部变量 `${Internal.Transformation.Filename.Directory}`，如果使用外部变量，要确保这些外部变量在运行时有正确的值。

如果在转换或作业里已经设置了变量，下面就是如何给这些变量赋值。可以使用kettle.properties文件。也可以用你自定义的文件，并在作业的第一个作业项里读取这个文件并设置变量。另外还可以使用其他方法，如使用参数表，使用数据库参数表更灵活，例如可以把类似开始日期、结束日期等这样的参数放在数据库表中，见表13-1的例子。

从参数表获取参数还有一个作用，很多其他ETL工具也使用参数表的方式保存参数，当企业使用Kettle来代替其他ETL工具的时候，已有的参数表可以不做修改，还继续作为Kettle的参数表。

表13-1 ETL参数表

| id | environment | prm_name | prm_value | validfrom | validto |
|----|-------------|----------|-----------|------------|------------|
| 1 | dev | dbhost | sagitta | 01/01/1990 | 12/31/2999 |
| 2 | tst | dbhost | virgo | 01/01/1990 | 12/31/2999 |

续表

| id | environment | prm_name | prm_value | validfrom | validto |
|----|-------------|----------|-----------|------------|------------|
| 3 | acc | dbhost | scorpio | 01/01/1990 | 12/31/2999 |
| 4 | prd | dbhost | aquarius | 01/01/1990 | 06/30/2010 |
| 5 | prd | dbhost | capricorn | 07/01/2010 | 12/31/2999 |
| 6 | dev | uname | usr123 | 01/01/2009 | 12/31/2999 |
| 7 | prd | uname | usr456 | 01/01/1990 | 12/31/2999 |

如何从上面的参数表获取参数呢？实际我们只要根据environment这一列以及当前的系统时间就可以获取需要的变量，变量是键—值对的形式。

说明：对键—值对的处理参考第20章。

使用表输入步骤从上面的参数表获取变量，表输入步骤里的SQL语句如下：

```
SELECT  prm_name
FROM    prm_value
FROM    kettle_param
WHERE   environment = '${environment}'
AND     validto      >= sysdate()
```

例如我们目前是在生产环境，我们先给environment这个环境变量赋值为“prd”，然后再执行表输入步骤，就可以抽取所有生产环境下的变量。

在生产环境执行完上述SQL后，会返回如表13-2所示的结果。

表13-2 参数查询结果

| prm_name | prm_value |
|----------|-----------|
| dbhost | capricorn |
| uname | usr456 |

有了生产环境的连接参数后，还要把这些参数通过“设置环境变量”步骤设置给变量。“设置环境变量”步骤可以读取字段数据，并把每个字段的数据设置为变量。但数据的格式必须如表13-3所示。

表13-3 环境变量的数据结构

| | |
|-----------|--------|
| dbhost | uname |
| capricorn | usr456 |

我们需要一个额外的步骤“Row denormalizer（反规范化/行转列）”步骤来完成这个工作。这个步骤可以把表13-2格式的数据转换为表13-3 格式的数据。图13-5 是这个步骤的设置窗口。

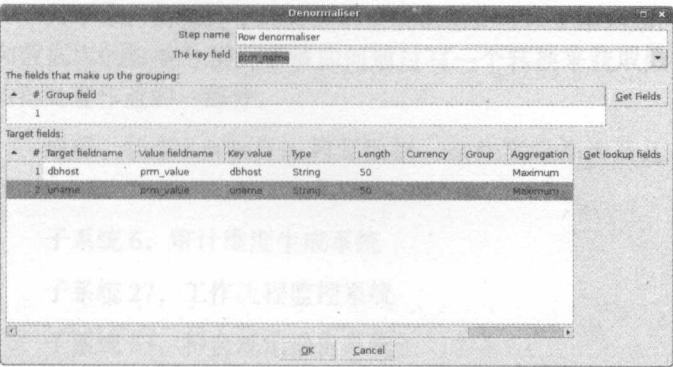


图13-5 “Row denormalizer（反规范化/行转列）”步骤的设置

从图13-5中可以看到，这里的转换不需要设置“Group”字段。图13-6显示了完整的转换，用来把数据库表里的参数值设置为Kettle 变量，转换包括三个步骤。

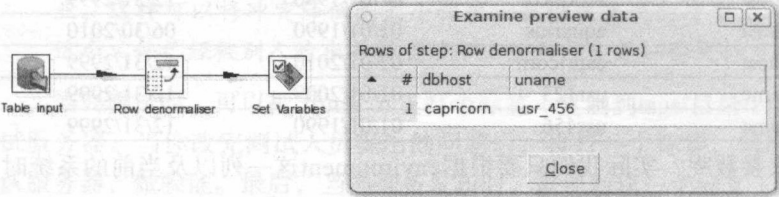


图13-6 完整的设置变量的转换

这个转换也可以从本书的例子里下载。记住，这个转换应该是一个独立的转换，它应该在作业里其他转换的前面执行。

13.5 小结

本章我们介绍了两个重要的ETL子系统，子系统25版本控制和子系统26版本移植。我们讨论了下面的主题：

- 什么是版本控制系统，如何在你的数据集成项目里使用版本控制系统。
- Kettle 转换和作业的元数据。
- Kettle 的不同资源库类型。
- 如何创建和升级Kettle 资源库。
- 把作业或转换从开发环境移植到测试、确认、生产环境的两个方法。
- 如何使用配置文件和参数表的方式使Kettle解决方案参数化。

第14章 血统和审计

当你创建了很多 Kettle 作业和转换后，你就会发现追踪结果是一件很困难的事情。另外，在作业发生了一些问题后，如何审计和判断问题的原因也是非常重要的一件事情。我们需要知道作业执行了什么、错误发生在哪里、执行一个作业要多长时间。本章为你讲述如何完成这些任务并且讲述血统分析、影响分析和审计等主题。

回忆一下第5章讲述的ETL子系统，子系统29 覆盖了血统分析和依赖分析。血统分析要分析结果是如何产生的，从过程的终点向后分析产生结果的各个步骤。而影响分析是从过程的起点向前分析对步骤和结果的影响。大致来说，影响分析是从源头来分析，而血统分析是从目标来分析。

对于影响分析和血统分析来说，你都需要元数据，本章前面部分将告诉你如何使用一个转换来读取元数据。这样你就可以自动抽取血统信息，也可以把抽取元数据的转换作为每天晚上跑的转换的一部分，甚至可以和第三方软件分享这些血统信息。

接下来，将描述不同种类的血统信息。可以了解到如何在 Spoon 里获取字段级别的血统分析和数据库的影响分析。了解如何通过写一个转换来获取数据库影响分析，并把这个转换作为你的批处理作业的一部分。

最后，讲解 Kettle 日志和元数据，这些实际都涉及几个 ETL 子系统，尤其是下面几个子系统：

子系统 6，审计维度生成系统

子系统 27，工作流程监控系统

子系统 33，符合规定报告系统

14.1 批量血统抽取

越来越多的应用系统加入到企业的应用平台中，要追踪这些系统里的数据流也变得越来越大。随着基于关系型数据库的应用系统的不断繁衍变化，传送数据的情况越来越多。像Kettle这样的数据整合工具只能在一定范围内跟踪最终用户数据的来源。也就是说，在一个大企业里，一个数据整合工具也只能负责一小部分数据传送的工作。所以，数据整合工具应该可以向掌握企业里所有数据流程的第三方系统报告。如果数据整合工具做不到这一点，就会有问题，就不是一个完整的解决方案。

为了解决Kettle 转换面临的这个问题，我们提供了一个转换例子，这个转换例子可以搜索一个目录下的一批转换里的所有表输出步骤，并报告其中用到了数据库的转换名和步骤名，以及这些转换里用到的数据库名和表名。

图14-1就是这个转换例子（文件名extract-transformation-metadata.ktr）。

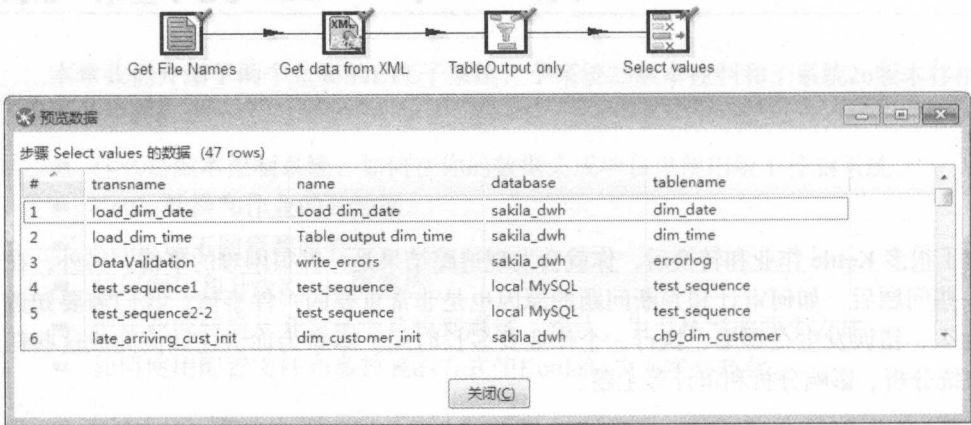


图14-1 抽取转换的血统信息

这个转换先从一个目录里读取一组转换文件名。使用通配符 *.ktr\$来匹配目录下所有转换文件。然后解析XML格式的转换文件，循环 /transformation/step节点。关于转换和作业XML文件的结构参考第13章。我们需要的信息都保存在XML节点里，使用下面的XPath 表达式就可以抽取到这些信息。

- 转换名: /transformation/info/name
- 步骤名: ./name
- 步骤类型: ./type
- 数据库连接名: ./connection
- 目标数据库表名: ./table

剩下要做的就是过滤出步骤类型是TableOutput的步骤，并保留想要的字段。

这个简单的例子说明为了做血统分析或影响分析而从XML格式的转换或作业文件中抽取Kettle元数据实际上是很简单的。

说明：如果你使用的是资源库，也可以有很多种方式把资源库导出成XML格式文件。导出方式参考第13章。

从Kettle 转换中抽取出元数据可以做很多事情。例如，抽取出的元数据可以保存在关系型数

数据库中，便于以后分析，如使用Pentaho的报表或分析套件来分析和展现。也可以通过元数据来监控ETL解决方案自身的质量。例如，可以看到一个转换或作业是否有适当的日志配置、是否有转换描述或注释。

在作业这个层次上，也可以抽取到一些有意义的元数据。下面有几个建议：

- 查询作业里的FTP 作业项，列出所有的FTP系统。这样你就可以获取你抽取的文本文件的来源。
- 检查邮件作业项的设置，看邮箱地址是否是硬编码的。
- 列出所有复制文件或删除文件的作业。

所有这些信息都可以从类似 `extract-transformation-metadata.ktr` 的转换中得到。唯一不同的是要从作业XML文件中读取，从不同的节点下读取。

14.2 血统

在转换中，血统就是说你要知道一个数据是从哪里来的，在哪个步骤中，增加了或修改了这个数据，最后输出到哪个数据库表中。本章覆盖了Kettle里的不同血统特性。

14.2.1 血统信息

在一个Kettle 转换里，在往输入步骤里添加字段时，一般遵循最小化影响的原则。就是说如果字段没有变化或没有被使用到，就不必加入到步骤中。这样将来维护的工作量最小。

Kettle的元数据结构不但可以让开发人员知道一个步骤有哪些输入字段和输出字段，而且可以知道一个字段是在哪里创建或修改的。

看一下前面的图14-1。打开转换（`extract-transformation-metadata.ktr`），右键单击“Get data from XML”步骤，选择“显示输入字段”命令。该命令将列出指定的步骤的所有输入字段。在这个例子里，将列出所有要抽取的文件名和其他文件相关属性列。在本例中，查询结果中的“步骤来源”列说明了“Get File Name”步骤是这些字段的来源步骤。如果在“Get data from XML”步骤的右键菜单里选择了“显示输出步骤”，就会发现多了几个额外的字段，这些字段的来源步骤就是“Get data from XML”步骤，如图14-2所示。

说明：当把鼠标放在一个步骤上，并按下空格键，也可以出现“显示输出步骤”对话框。

因为在“TableOutput only”步骤里字段没有被移除和修改，所以这个步骤的输入字段和输出字段一样。从第一个步骤获得的文件名将一直往下面的步骤传递，直到遇到了“Select values”步骤，这个步骤会移去不再需要的字段。

这种把字段向下传递的规则可以使开发人员很容易地改变字段名或添加步骤，而不影响其他步骤。但是在一些步骤里你也需要告诉Kettle这些字段的用途，如在“Get data from XML”步骤里，你需要指定要在 `filename` 字段里读取文件名。在大多数步骤里，都可以很方便地通过列表、下拉按钮等方式来选择字段名。

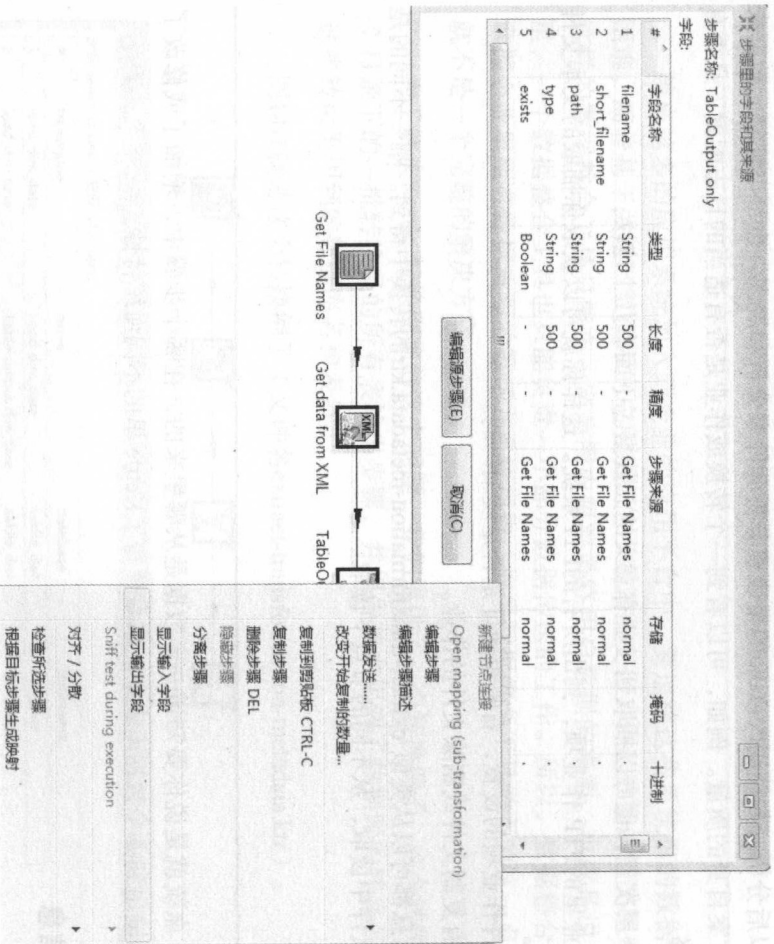


图14-2 获取步骤的输出字段

14.2.2 影响分析信息

有时候需要知道一个转换里哪些地方使用了数据库，一个转换的两个不同步骤里是否使用了同一个数据库表。Spoon 里提供的数据库影响分析列出了一个转换里所有的这些信息。图14-3显示了把数据加载到事实表的影响分析结果。

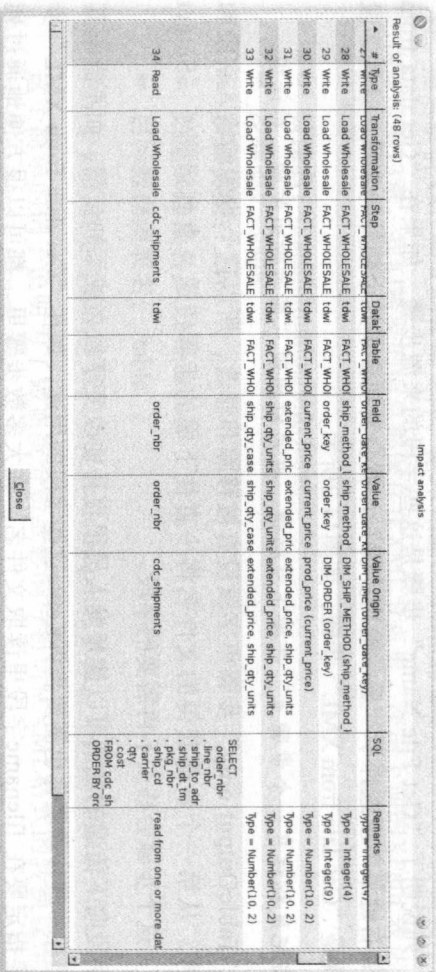


图14-3 显示数据库影响分析的相关信息

上面的表里列出的信息是从转换的每个步骤里抽取出来的。每个使用了数据库连接的步骤都在这个表里有对应的数据行。如果可能，这些信息将会细到字段的级别。

如果你不想通过用户界面的方式获得这些信息，也可以使用API的方式调用，这些功能都已经以API的方式实现了。通过API，可以充分发挥开发人员的想象力。为了简化工作，也可以使用几行简单的JavaScript（参考db-impact.ktr例子）：

```
var transMeta = new Packages.org.pentaho.di.trans.
    TransMeta(filename);
var transname = transMeta.getName();
var impact = new Packages.java.util.ArrayList();
var error = "";
```

```
try {
    transMeta.analyseImpact(impact, null);
} catch(e) {
    error = e.toString();
}
```

```
for (i=0;i<impact.size();i++) {
    var dbi = impact.get(i);
    var newRow = createRowCopy(
        getOutputRowMeta().size());
    var rowIndex = getInputRowMeta().size();
```

```
    newRow[rowIndex++] = transname;
    newRow[rowIndex++] = dbi.getStepName();
    newRow[rowIndex++] = dbi.getTypeDesc();
    newRow[rowIndex++] = dbi.getDatabaseName();
    newRow[rowIndex++] = dbi.getTable();
    newRow[rowIndex++] = "N";
    newRow[rowIndex++] = error;
```

```
    putRow(newRow);
}
```

```
var ignore = "Y";
```

这段脚本加载转换元数据并使用Kettle的Java API来抽取各种影响信息，如数据库类型、转换名、步骤名、数据库名、表名等。这段脚本还可以处理在做影响分析时发生的错误。

这段脚本对输入的每个转换文件都生成N行影响分析的结果。最后脚本使用了putRow()方法将生成的结果写到步骤的输出中。关于putRow()方法可参考第23章。

这些获取到的信息可以放在报告中或存在一个数据库里便于将来分析使用。例如，可以列出使用了某个数据库表的所有转换，等等。当我们要对某些表进行修改时，就需要这样的功能来查找对哪些转换产生了影响。

图14-4 显示了例子里的几行输出。

浏览数据

步骤 Filter rows 的数据 (76 rows)

| | transname | stepname | type | db | table | ignore | err |
|---|---------------|----------------|------|--------|---------|--------|-----|
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup Address | 读 | sakila | address | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup City | 读 | sakila | city | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup City | 读 | sakila | city | N | |
| 79_downloads\635179_code_ch04\fetch_address.ktr | fetch_address | Lookup City | 读 | sakila | city | N | |

关闭(C)

图14-4 数据库影响分析

14.3 日志和操作元数据

对于ETL的开发和运维人员来说，经常需要获取有用的日志信息。在本节，我们了解一下Kettle的日志架构和配置方式。

14.3.1 日志基础

Kettle里的大部分组件都能够以文本行的方式输出日志信息。例如当步骤结束时，可以生成下面的一行日志：

```
2010/06/18 10:36:29 - Step name.0 -  
Finished processing (I=0, O=0, R=0, W=25, U=0, E=0)
```

这个日志行里有以下主要内容：

- 日期和时间。
- 步骤名，步骤名的后面有一个点号，点号后面是步骤的拷贝号。
- 日志内容。
- 每个部分之间都使用空格、横线、空格来分割。

当你执行一个转换或作业时，你需要选择日志级别。根据你选择的日志级别的不同，输出的日志行数也不同。Kettle 里有下面几个日志级别。

- 行级日志：打印出 Kettle 里所有可用的日志信息，包括在一些比较复杂的步骤里的每一行数据。
- 调试日志：也打印出很多日志信息，但没有到每一行数据都打印出来的级别。
- 详细日志：比基本日志多一些日志内容，如多了SQL查询语句和DDL 语句等。
- 基本日志：默认的日志级别，只打印出执行到哪个步骤和作业项的日志信息。
- 最小日志：只打印出执行到哪个转换和作业的日志信息。
- 错误日志：只打印出错误日志，如果没有错误不打印日志。
- 无日志：不打印任何日志，即使有错误也不打印。

可以在Spoon的转换或作业的运行对话框里设置日志级别。当使用Pan命令或Kitchen命令时，也可以在命令行里通过 -level 参数指定。最后，还可以在转换或作业的执行结果面板里来设

置默认日志级别。在执行结果面板的“日志”标签下单工具条里的相应图标（螺丝刀和扳手图标），打开如图14-5的对话框。

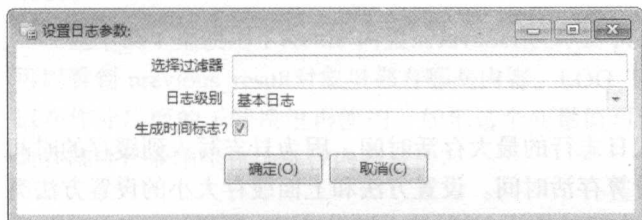


图14-5 设置日志参数

设置日志参数对话框里一个很有用的功能就是可以指定一个过滤字符串。指定过滤字符串后，只有包括指定字符串的日志行才会被保留下来。在详细日志里可以通过这个功能，来过滤出带有特定值的日志行。例如，若你指定了步骤名，只有包含这个步骤名的日志行才被保留在日志里。

14.3.2 日志架构

自从Kettle 版本4 以后，所有的日志行都被保留在一个中心缓存中。这个缓存并不只是简单保存上节所说的日志的文本内容。实际上中心缓存里保存了以下内容。

- **日期和时间**：让Spoon在日志窗口用蓝色显示日志的日期和时间。
- **日志级别**：让Spoon在日志窗口用红色显示错误行。
- **递增的唯一编号**：Kettle对不同日志窗口或远程日志抽取分配的递增的编号。
- **日志文本内容**：实际的日志文本内容。
- **日志通道ID**：这个ID是随机生成的唯一字符串，用来唯一标识生成日志行的Kettle组件。

把日志保存在内存里是导致内存溢出的一个主要的原因。当你执行一个作业或转换而且日志级别设置得比较高时，在执行过程中就会产生大量日志。如果你要求Kettle 把这些日志信息存储在日志表里，那么这些日志信息就会暂时保持在内存里，最后就会发生内存溢出问题。另外ETL开发人员也不可能看几十万行的日志信息，通常在错误发生时，开发人员会关心最后的几千行日志。基于上面的原因，同时还有第18章实时数据整合描述的一些原因，一般都要限制保存在中心日志缓存中日志的数量。限制日志的参数将在下节介绍。

设置缓存大小

限制日志数量通常想到的第一个方法就是设置缓存大小。可以在Spoon的“选项”对话框里设置这个参数。在对话框里的“日志窗口的最大行数”输入框里设置。要注意该选项只在Spoon里有效。

另一个选择就是设置 `KETTLE_MAX_LOG_SIZE_IN_LINES` 环境变量。可以在kettle.properties文件里设置这个环境变量（参考第18章实时数据整合实现的日志部分）。

最后，也可以在Carte的配置文件中设置这个参数。例如在服务器的配置XML文件中加入下面的配置行，将中心日志缓存的大小设置为10 000行：

```
<!-- Prevent out of memory by only keeping 10,000
      rows in the central log buffer
-->
<max_log_lines>10000</max_log_lines>
```

设置日志行最长保留时间

一个解决内存溢出更好的方法是设置日志行的最大存活时间。因为日志写入到缓存的时间是知道的，我们就可以根据这个时间来计算存活时间。设置方法和上面缓存大小的设置方法类似，在“选项”对话框的“内存中保留日志行时长(分钟)”输入框里设置。同样要记住这个选项也只对在Spoon里运行的转换有效。

定义日志行最长保留时间的参数是KETTLE_MAX_LOG_TIMEOUT_IN_MINUTES。在Kettle 4以后的版本里可以直接在Spoon的“编辑kettle.properties文件”菜单项里编辑kettle.properties文件。

也可以在Carte的配置文件里加入下面的内容来设置日志的最长保留时间：

```
<!-- Discard log lines if they are in the
      buffer for more than 2 days -->
<max_log_timeout_minutes>2880</max_log_timeout_minutes>
```

日志通道

在Kettle 4 以前，所有的日志文本都被写到日志后台（见 Apache Log4J, <http://logging.apache.org/log4j>）。这种方式会引起很多问题，如作业和转换在一台机器上并发执行时各自的日志内容会混在一起，把日志保存在内存里还会造成巨大的内存开销。日志记录后，也很难过滤出属于某一个步骤或数据库的日志。

为解决这个问题，在Kettle 4 版本里创建了日志缓存机制，这在前面的“日志架构”中已经描述过。当执行一个作业或转换时，每个组件（作业项、步骤、转换和数据库连接）都会创建一个分离的日志通道，每个日志通道都有一个唯一的ID，这样可以知道日志产生于哪个组件。因为Kettle 也能追踪到组件的父日志通道ID，所以作业执行的层次关系也被记录下来。因为这种层次关系是分开保存的，所以Kettle 也能追踪到某个日志行属于哪个组件及它的子组件。

例如，假设你在Spoon 里执行了一个作业，这个作业里有一个长时间运行的转换正在运行。你能看到当前运行的作业项上有一个小蓝图标，可以右键单击这个作业项打开相应的转换，这种方法也可以用于子作业。这时会显示出转换（作业），就像你独立启动一个转换，可以获取当前转换运行时的各种度量值和日志信息，而不是它的父作业的日志信息。这就是中心日志缓存和层次日志通道架构。

在一个作业里捕获日志

从Kettle 4以后，可以在一个作业项的运行结果里找到这个作业项的运行日志。通过JavaScript 作业项就可以从结果里抽取日志内容。尽管JavaScript 作业项的主要目标是评估一个布尔值表达式是true还是false，不过这个作业项也可以被用来抽取前一个作业项运行结果里的不同内容。下面的脚本里设置了一个变量，变量的值就是上个作业项（转换或作业）的日志：

```
var logText = previous_result.getLogText();
```

```
parent_job.setVariable("LOG_TEXT",logText);  
  
true;
```

这个例子里使用了Kettle 内置的JavaScript对象 previous_result。在第22章的“结果”一节里可以看到 previous_result对象里都有哪些内容。LOG_TEXT 变量定义在了父作业里，这个变量可以在作业后面的子转换里再使用，如把这个变量值写到数据库里，或者保存成XML文件等。或者在邮件作业项里作为邮件内容发送。

14.3.3 日志表

因为从远程服务器上搜索日志文件并不方便，所以 Kettle 提供了把日志写入到日志表的功能。在下面部分我们看看Kettle可以把哪些内容写入到日志表。这里要强调一下，把日志写入日志表是一个好的习惯，这样便于追踪和调试错误。

转换日志表

转换有四个日志表：

- 转换日志表
- 步骤日志表
- 性能日志表
- 日志通道日志表

在默认情况下，转换运行结束后日志被写入到事先配置好的响应的日志表中。转换日志表是一个特例，你会发现，在转换开始执行时就有一条记录被写入到日志表中。在转换开始执行时，Kettle 需要为转换确定一个批 ID 号，这个号对每次执行的转换都是唯一的，可以用这个批 ID 号在转换日志表里做分组操作。当转换正在运行时，也可以使用获取系统信息步骤来获取本次运行的批 ID 号，这样可以通过这个号从日志表里查询相关内容。

配置好日志表后，可以在“执行结果”面板的“执行历史”标签下检查执行的结果，如图 14-6所示。如果你没看见“执行结果”面板，单击转换工具条上的放大缩小百分比左侧的那个图标，来显示“执行结果”面板。

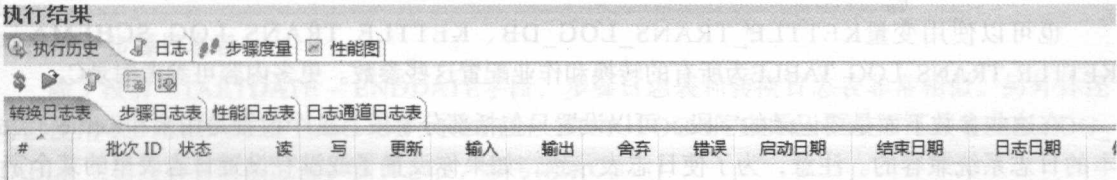


图14-6 转换执行历史

转换日志表在转换的“设置”对话框里配置。该对话框可以通过Spoon 菜单条的“编辑→设置”菜单命令打开，也可以右键单击转换的背景选择“转换设置”菜单命令。

使用对话框里的SQL 按钮，可以按照你配置好的日志表的字段生成对应的SQL 语句。注意在编写本书时，日志表的索引还没有自动创建。下面有关于添加索引的一些建议。

转换日志表

转换日志表用来保存写到数据库里的转换过程的度量信息。图14-7 显示了可以设置的关于转换日志表的所有属性。

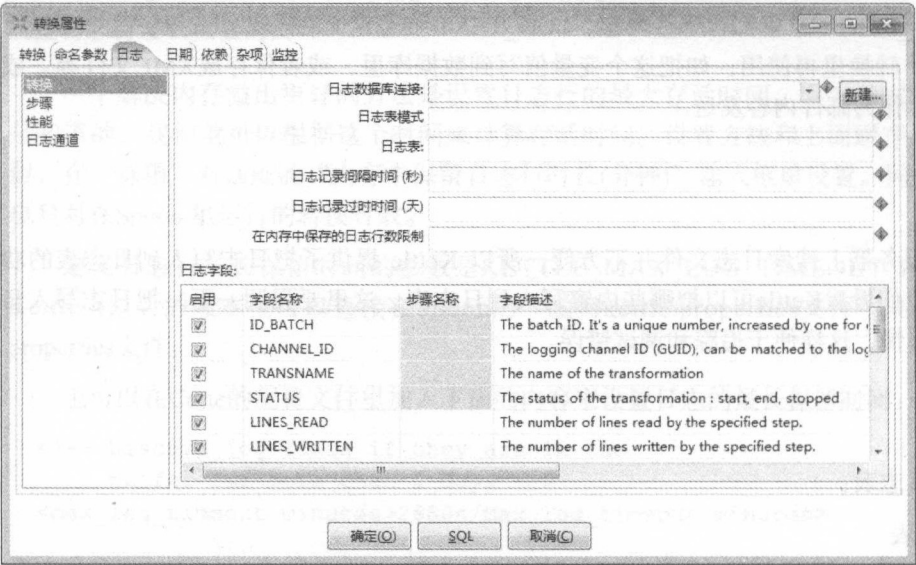


图14-7 转换日志表设置

在标签中的左侧，是转换里不同的日志表的名字。下面列出了可以设置的参数。

- 日志数据库连接：包含日志表的数据库连接。
- 日志表模式：包含日志表的模式。
- 日志表：日志表的名字。
- 日志记录间隔时间(秒)：如果为这个参数设置了值，转换将按照设置的间隔时间往日志表里写信息。如果不设置，日志表只在开始时更新一次，在结束时更新一次。
- 日志记录过时间（天）：这个日志参数将从日志表中移去比超时天数早的日志记录。使用日志表里的日期字段来做判断。
- 在内存中保存的日志行数限制：对于不支持太大的文本字段的数据库，这个选项可以限制日志文本的行数。

也可以使用变量KETTLE_TRANS_LOG_DB、KETTLE_TRANS_LOG_SCHEMA、KETTLE_TRANS_LOG_TABLE为所有的转换和作业配置这些参数。更多内容可参考附录C。

在这些参数下面是要记录的字段，可以设置只包括部分字段。这个设置是用来和Kettle 3 版本的日志系统兼容的。注意，为了使日志表一致，如果你设置了或没有设置日志表里的某个字段，你需要在其他转换里也做同样的设置。因为这样设置起来非常烦琐，所以最好保持Kettle 默认的日志设置。另外，也可以给日志表里的字段改名，但实际很少会这么做。

在“步骤名称”这一列，可以指定一个步骤作为日志里的某个度量值，如“写”行数。这样整个转换的所有度量值可以通过不同的步骤来表示。例如你在步骤A里读取一个文本文件，在步骤B里把数据写到数据库表。此时就应该把 LINES_INPUT字段设置为步骤A，把 LINES_OUTPUT字段设置为步骤B。这样，可以从整体上了解一个转换处理了多少行数据。如果你不指定任何字段，在转换日志表里这些字段对应的值都是0。

“字段描述”列说明了日志表里每个字段的用途，解释了一些容易混淆的字段名。例如，STARTDATE这个字段实际并不是转换的开始时间。实际这个字段是用于增量数据处理时的开始时间。关于增量数据处理参考第6章的“变更数据捕获”一节。这一节里讲了几种捕获变更数据的方法。除了推荐的cdc_time表的方法，另外就是使用日志表里的STARTDATE字段的方法。

STARTDATE – ENDDATE这两个时间点之间的时间片段里包括了自从上次转换运行后系统里新增加的变化的数据。这两个时间值是在转换刚启动时就计算好并记录到日志表里的，可以使用“获取系统信息”步骤获得这两个时间点。

如果你觉得日志表以后会增长得比较快、比较大，就应该给日志表建立索引。这将加快转换对日志表的查询和更新。首先要在 ID_BATCH 列上建索引，然后要在ERRORS、STATUS和TRANSNAME 列上建索引。

步骤日志表

因为转换日志表是在一个比较高层次上的日志，通常用户还希望可以看到步骤级别的日志，每个步骤处理了多少行数据。同样在转换的属性对话框里，在“日志”标签下来配置步骤日志表，如图14-8所示。

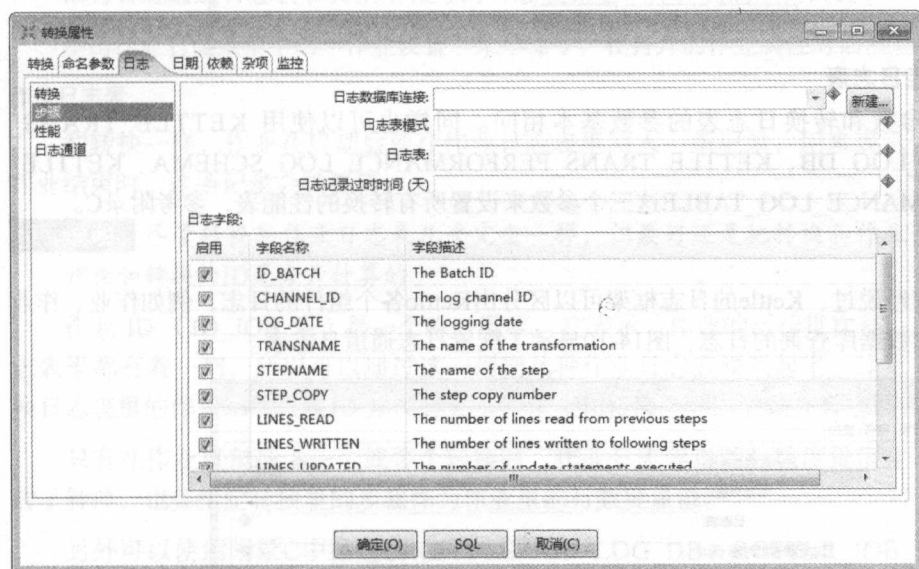


图14-8 步骤日志表

除了没有 STARTDATE – ENDDATE字段，步骤日志表和转换日志表非常相似。另外你还可以使用 KETTLE_STEP_LOG_DB、KETTLE_STEP_LOG_SCHEMA和KETTLE_STEP_LOG_TABLE 变量为所有转换配置步骤日志表，参考附录C。

如果你还想记录步骤级别的日志信息，可以选中 LOG_FIELD字段。因为这个字段不常用，所以默认情况下，步骤日志表里不包括这个字段。

性能日志表

第15章将讨论如何开启性能监控，通过性能监控可以看到每个步骤的性能情况。性能监控需要定时抓取所有步骤的度量值。这些信息在转换运行阶段通过变化的图形展现出来。

这些信息对性能分析很重要，性能问题不仅在开发阶段存在，当批量运行完一组转换后同

样需要这些信息来分析性能问题。这就是性能日志表存在的原因。图14-9 显示了性能日志里的所有选项。

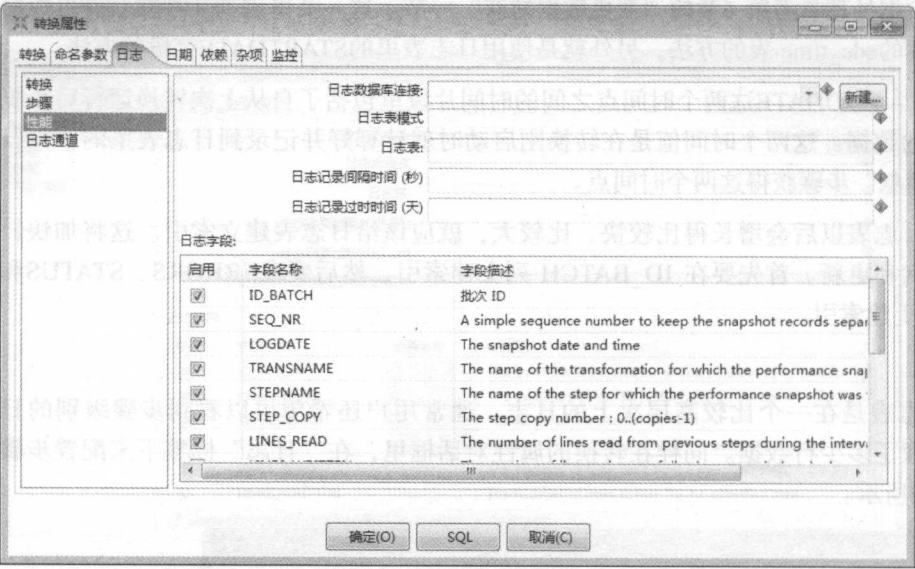


图14-9 配置性能日志表

这里列出的参数和转换日志表的参数基本相同。同样也可以使用 KETTLE_TRANS_PERFORMANCE_LOG_DB、KETTLE_TRANS_PERFORMANCE_LOG_SCHEMA、KETTLE_TRANS_PERFORMANCE_LOG_TABLE这三个参数来设置所有转换的性能表，参考附录C。

日志通道日志表

在本章前面曾经说过，Kettle的日志框架可以区分出Kettle各个组件的日志，例如作业、作业项、转换、步骤和数据库查询的日志。图14-10显示了配置日志通道日志表。

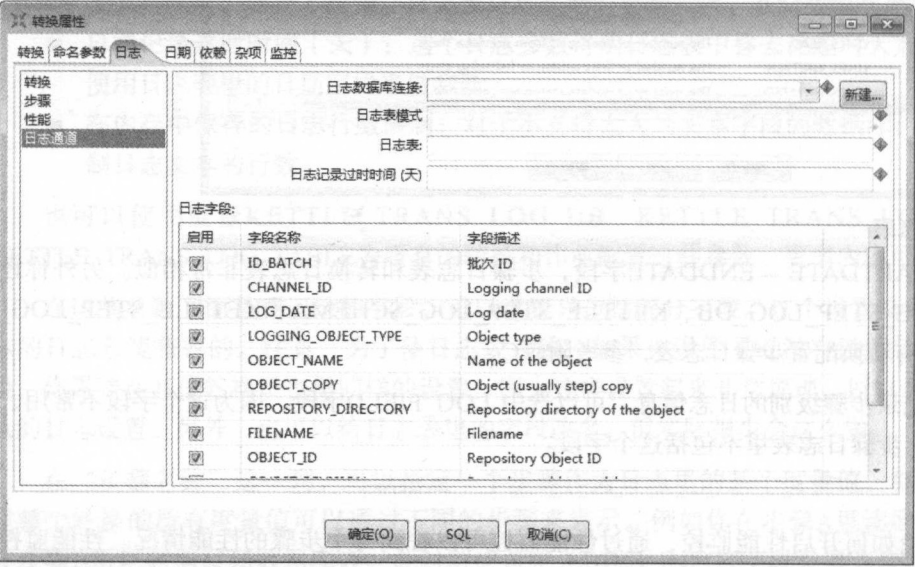


图14-10 配置日志通道日志表

数据库连接、模式、表的配置和前面其他日志表的配置方法相同。另外可以使用附录C

中描述的KETTLE_CHANNEL_LOG_DB、KETTLE_CHANNEL_LOG_SCHEMA、KETTLE_CHANNEL_LOG_TABLE变量让所有的转换，使用同一个日志表。

下面我们看看用这些日志信息能做哪些事情：

- 可以列出作业里的哪些转换被执行了。
- 当一个转换失败后，找到这个转换的准确版本号。
- 找到哪个最近运行的转换使用了某个映射步骤（可能有人把这个映射步骤写错了，需要找到哪个转换受到了影响）。

作业日志表

因为没有作业性能监控的数据，所以作业只有下面三种日志表：

- 作业日志表
- 作业项日志表
- 日志通道日志表

因为日志通道日志表和转换里相应的日志表完全一样，我们不再讲述。

单击作业右键菜单中的“作业设置”菜单命令，在打开的作业属性对话框里设置作业日志表。

作业日志表

和转换一样，作业在启动后就往作业日志表里写入一条记录，用来表示作业开始执行。当作业结束时，这条记录会被更新。

说明：尽管转换和作业日志表几乎完全一样，但最好还是把转换和作业日志表分开，因为作业和转换的ID是分开计算的。

作业 ID（ID_JOB 列）是一个唯一数字，它表示了作业的运行批次号。因为在几种作业日志表里都有着一列，所以可以通过这一列把几种作业日志表联系起来。注意，作业日志表和转换日志表里的作业ID和转换ID 没有关系，作业和转换之间的关系在日志通道日志表里体现。

只有在作业里包括了一个或多个转换时，作业日志表里的行数度量值才不会是零。即使包含了转换，也要指定转换里的步骤作为作业里的行数度量值。

另外可以使用附录C中描述的KETTLE_JOB_LOG_DB、KETTLE_JOB_LOG_SCHEMA、KETTLE_JOB_LOG_TABLE 变量来为所有作业配置同一个作业日志表。

还要注意一下，最好给作业日志表建立两个索引来加快查询和更新的速度，第一个要建立的索引是ID_JOB列，第二个要建立的索引是ERRORS、STATUS和JOBNAME 列。

作业项日志表

在作业项日志表里，也能发现很多和转换日志表里相似的参数。主要的区别是作业项日志表里包括了作业项的运行结果。运行结果包括了布尔类型的结果标识字段、结果行数字段和结果文件数字段。

可以使用附录C中描述的KETTLE_JOBENTRY_LOG_DB、KETTLE_JOBENTRY_LOG_SCHEMA、KETTLE_JOBENTRY_LOG_TABLE 变量来为所有作业配置同一个作业项日志表。

14.4 小结

想知道在一个复杂的Kettle 转换或Kettle 作业里发生了什么事情并不容易。本章讲述了如何分析Kettle 里的数据流和解决问题，重点讲述了下面几点内容：

- 批量抽取血统信息。
- 如何在设计阶段，Spoon 用户界面里获取血统信息。
- 抽取数据库影响信息。
- 为转换和作业建立日志表形式的审计信息。

第四部分：性能和扩展性

本部分包括

第15章 性能调优

第16章 并行、集群和分区

第17章 云计算中的动态集群

第18章 实时数据整合

第15章 性能调优

本章深入讲解Kettle的性能调优。我们主要讲转换的性能调优，简单介绍一下作业的性能调优。

为方便对转换引擎感兴趣的读者，本章的第一部分先提供了关于转换引擎的几个详细的例子，来说明转换引擎的工作原理。你知道了转换引擎如何工作后，我们会聚焦到如何识别性能瓶颈上。然后就是一些建议，如何提高你的转换和作业的性能。

说明：Kettle新手可以跳过本章，直到遇到了性能问题再来阅读本章。在那个时候可以直接看本章，并学习到如何识别和解决遇到的性能问题。

15.1 转换性能：找到最弱连接

转换性能调优在概念上非常简单。正如其他任何的网络，就是要找到网络里最弱的一条连接。在一个转换里，就是要找到是哪个步骤引起了性能问题。为了进一步理解，我们看一个例子。下面的转换从数据库里读数据并写到另一个数据库里，如图15-1所示。窗口下半部分显示的就是转换执行过程中的性能度量值。

这个例子的工作原理是：Customers步骤从数据库中读取记录，并把每一行记录写入到两个步骤中的行集（Row Set）缓存中。Load customers步骤从缓存中读取这些记录。在本例中，下面两个因素将转换整体的性能。

- **Load customers步骤比较慢：**这是经常会发生的一种情况，因为在大部分情况下，写数据的速度比读数据的速度慢。在这种情况下，两个步骤间的行集缓存将会被填满。一旦缓存被填满，就不会接收来自Customers步骤的任何数据行。就是说Customers步骤不得不停下来以等待缓存有空间。导致的后果就是Customer步骤也会慢下来，其速度将会和Load customers步骤的速度趋向一致。

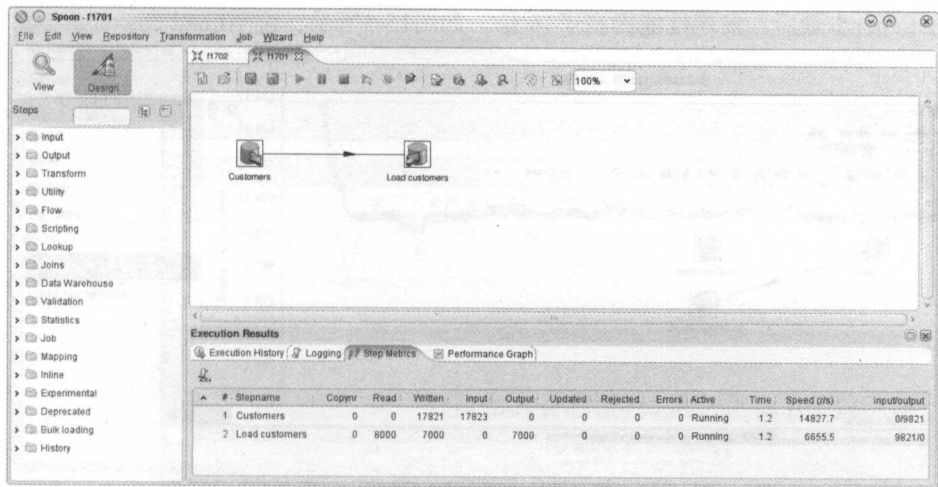


图15-1 读写数据

- Customer步骤比较慢：这种情况可能是因为读数据的网络比较慢，或数据库本身比较慢。在这种情况下，数据行写到行集缓存的速度很慢。“Load customers”步骤会很快消耗缓存里的数据，然后就等待直到缓存里有新的数据。和上面的情况相同，这两个步骤的速度也会逐渐接近。

上面两个步骤转换的原理同样也适用于任意多个步骤的转换，最慢的步骤影响了转换的整体性能。正因为如此，找到最弱连接将成为提高转换性能的最重要一步。

找到最弱连接或者说找到最慢步骤，可以有两个基本方法：简化或度量。下面分别说明这两种方法。

15.1.1 通过简化找到性能瓶颈

找到最慢步骤，通常使用的一种方法就是简化。将一个或多个步骤从转换中移除来观察哪一次移除可以最大程度提升性能。在上面的例子中，可以把转换改成如图15-2所示的样子。

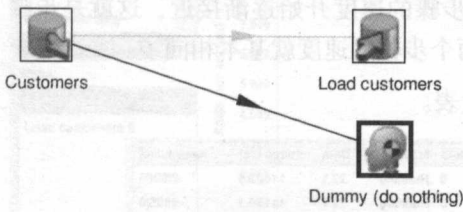


图15-2 移去一个步骤

如果你运行这个转换，可以发生两种情况：

- 如果性能提高了，可以知道Load customers步骤就是最慢的步骤。
- 如果性能没有提高，可以知道Customers步骤是最慢的步骤。

也可以通过使步骤之间的连线失效（在连线上单击鼠标），来运用这种方法。只有一个步骤和其他步骤之间的连线是有效的情况下，在运行时这个步骤才会被加入到转换中。通过这种连线失效的方法，可以快速排除大部分步骤，而把注意力集中到少数几个步骤上。

通常先在转换的后面开始使步骤间的连接失效，可以逐步观察哪个步骤是瓶颈。本例中，

当去掉了Load customer步骤后，发现性能提高了16 倍，如图15-3所示，说明Load customer步骤是性能瓶颈。

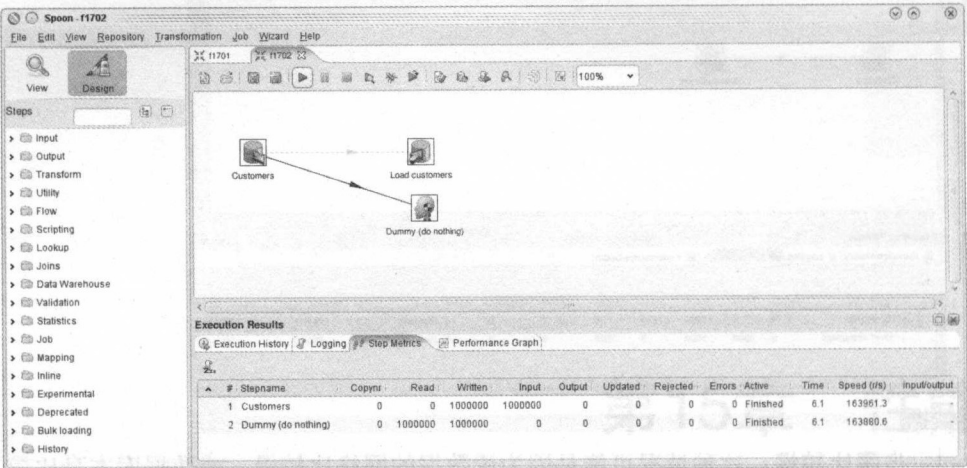


图15-3 执行结果显示性能提高

15.1.2 通过度量值找到性能瓶颈

另一个找到性能瓶颈的方法就是观察执行过程中缓存大小的变化。如果你执行了本章的第一个例子，在转换刚开始的时候，可以看到图15-1里窗口下方的性能度量表。

说明：当转换是运行在远程的Carte 上或PDI Server 上，可以在Spoon中，从服务器节点的右键弹出菜单中选择“监控”来查看远程执行的性能。

性能度量表的最后一列是input/output，实际就是一个步骤的输入和输出缓存内的记录数。在我们的例子里，设置的最大缓存大小是10 000，而我们有9821行记录在缓存里，就是说缓存的使用率是98%。这说明Customers步骤有足够的速度来填满缓存，反过来就是说，Load customers步骤不能快速消耗缓存里的数据，Load customers步骤就是性能瓶颈。

当转换运行一段时间后，性能度量开始发生变化。步骤的速度开始逐渐接近，这就是步骤之间的行集缓存导致的。如果缓存满了，一段时间以后两个步骤的速度就基本相同了。

在一段时间以后，可以看到如图15-4所示的性能度量表。

| # | Stepname | Copynr | Read | Written | Input | Output | Updated | Rejected | Errors | Active | Time | Speed (r/s) | input/output |
|---|----------------|--------|--------|---------|--------|--------|---------|----------|--------|---------|------|-------------|--------------|
| 1 | Customers | 0 | 0 | 256806 | 256808 | 0 | 0 | 0 | 0 | Running | 22.1 | 11603.9 | 0/9806 |
| 2 | Load customers | 0 | 247000 | 246000 | 0 | 246000 | 0 | 0 | 0 | Running | 22.1 | 11160.3 | 9806/0 |

图15-4 转换运行22秒后的性能度量表

尽管有性能度量表，但性能度量表是动态变化的，所以对于复杂的转换，想找出性能瓶颈仍然是比较困难的。这时使用转换的性能监控可以帮助我们找到瓶颈。

说明：步骤性能监控在“转换设置”对话框的“监控”标签里设置。可以看到每个步骤的性能图。在执行过程中性能监控会采集性能的度量值。

如果你看到了步骤性能度量，可以接着看步骤性能度量图，如图15-5所示。可以看到步骤的读写行数和缓存大小。在图15-5中，可以清楚看到Customers步骤最开始的速度很快，但后来就下降到和另一个步骤一样快的速度。

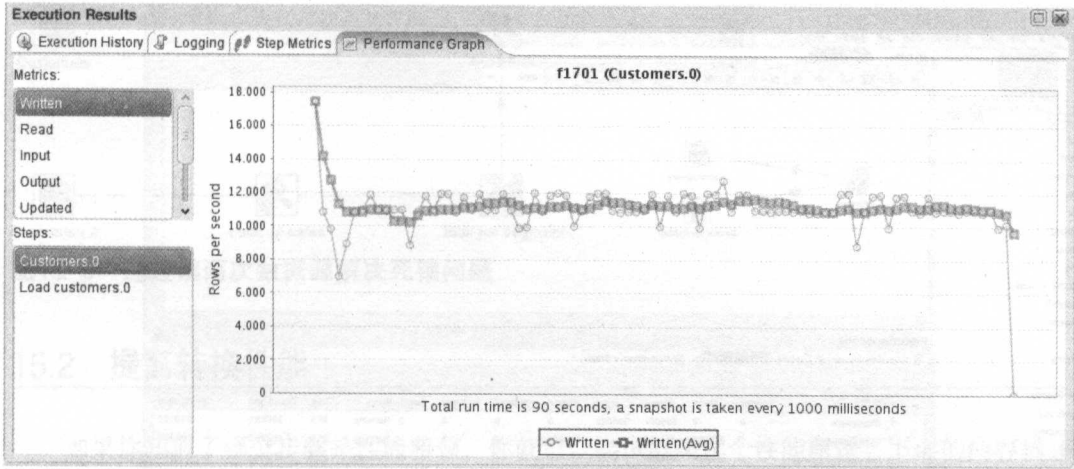


图15-5 步骤的性能度量图

图15-6 显示了在这段时间内缓存大小的变化。可以看到，缓存快速地被填满，然后一直保持这种状态直到转换结束。这种情况说明步骤间存在比较大的性能差距。（正如前面提到的，第一个步骤快16倍）。如果速度的差距比较小，缓存里的行数只会缓慢增长。

说明：在做性能测量时，在计算机内尽量不要运行其他程序。想做到这一点并不容易，邮件和Twitter客户端、病毒检查程序、垃圾邮件过滤程序、防火墙、浏览器等都会消耗一定的系统资源。

为了保证度量结果不受其他程序影响，要多次运行同一个转换以获得平均测量数据。当比较性能指标时，还要记住产生性能指标的机器要有相同的配置。即使处理器、磁盘或网络的速度有微小的不同，也会导致完全不同的性能测量结果。

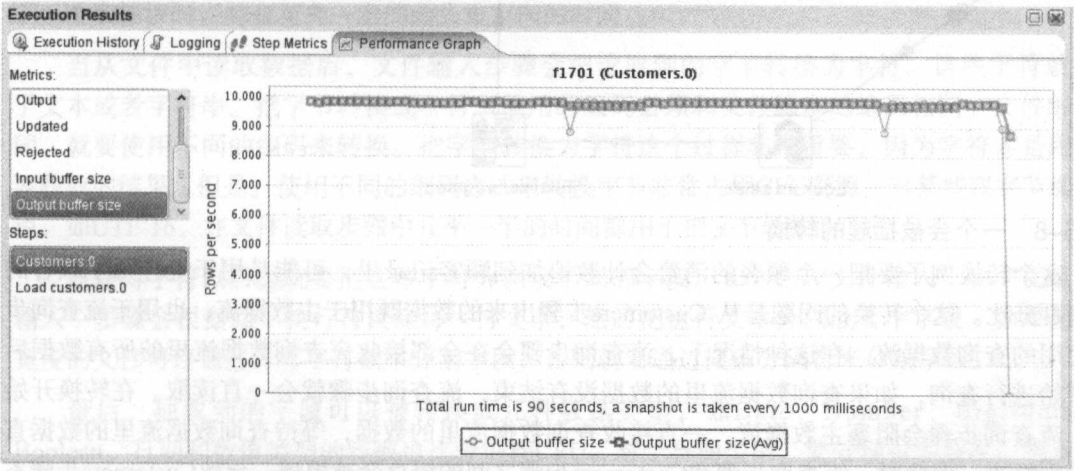


图15-6 一个步骤的输出缓存大小的变化图

15.1.3 复制数据行

一种比较特殊的性能调优的情况就是数据行被复制到多个目标中。在图15-7中的例子中，数据行被复制到一个非常快的do nothing步骤和Load customers步骤。

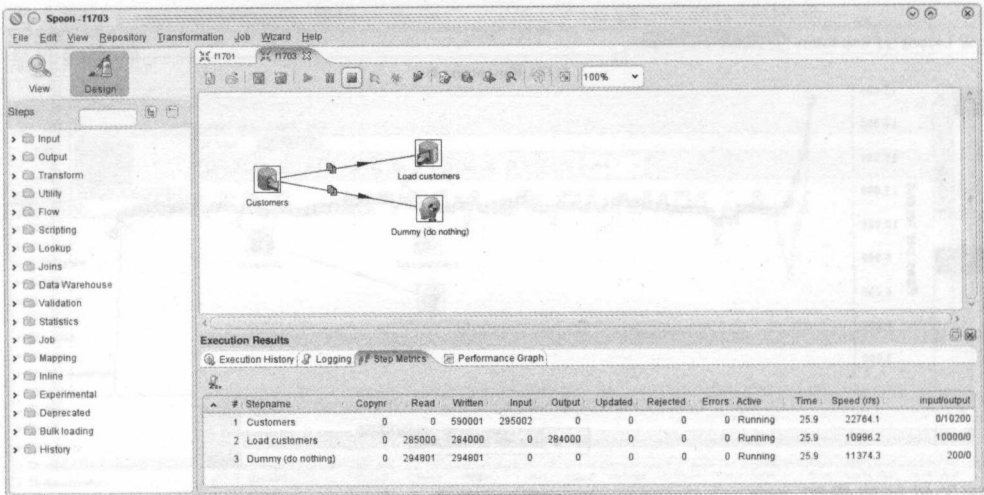


图15-7 复制数据行到多个目标步骤

注意尽管do nothing步骤的输入缓存几乎总是空的，Load customers步骤的性能会减慢do nothing步骤的速度。do nothing步骤的速度减慢是因为Customers步骤有两个输出缓存，只要其中一个输出缓存是满的，Customers步骤就会等一段时间再输出，导致do nothing步骤的速度会降低。

还有一种不太常见但是很重要的情况：一个步骤可以导致整个转换被挂起，看如图15-8所示的这个例子。

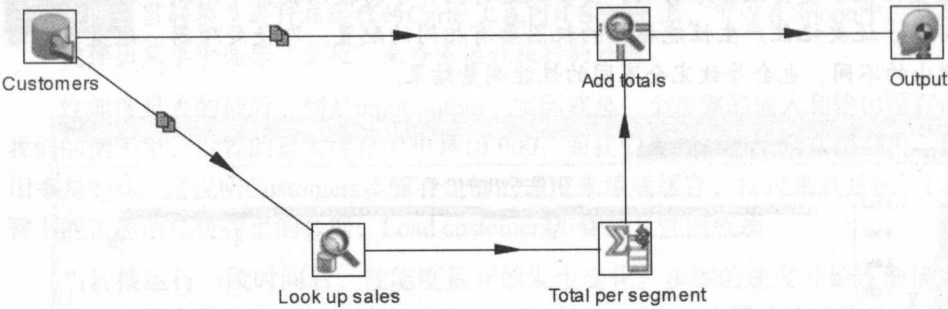


图15-8 一个会被挂起的转换

这个转换例子要把一个顾客的消费合计数追加到顾客记录里，可能是用于统计每个顾客的消费百分比。这个转换的问题是从 Customers步骤出来的数据既用于主数据流，也用于流查询步骤使用的查询数据流。在这种情况下，流查询步骤会在全部接收完查询数据流里的所有数据后才开始进行查询，如果查询数据流里的数据没有结束，流查询步骤就会一直读取。在转换开始后，流查询步骤会阻塞主数据流，一直接收查询数据流里的数据，等待查询数据流里的数据直到数据结束，而此时，流查询步骤和Customers步骤之间的缓存会被填满，导致Customers步骤不能再继续发送数据给查询数据流。这样整个转换就进入了死锁状态，两个或多个步骤在互相等待。

尽管可以通过加大步骤间的缓存解决这个问题，但是最好的方法还是避免循环引用。可以把转换改成图15-9的样子，让转换读取数据源两次。

也可以把一个转换分成两个转换，把查询数据临时保存在临时文件或数据库表中。

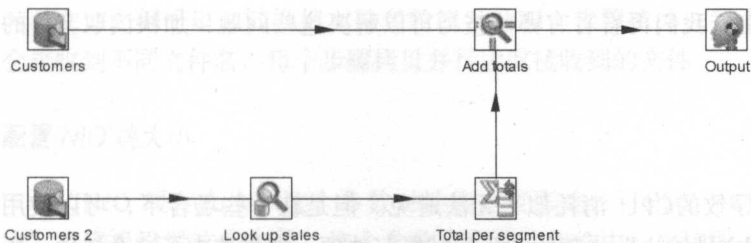


图15-9 通过读两次数据源解决死锁问题

15.2 提高转换性能

如果你知道了哪个步骤是性能瓶颈，你就可以尝试解决这个性能瓶颈。下面的内容描述了几种使转换跑得更快的方法。首先，你要知道如何提升读写文本文件的性能。然后，要知道如何提升读写数据库的性能。因为网络的性能也影响了数据库的读写性能，我们也会介绍一些如何提升网络性能的主题。在本节的最后，再针对某些特定的步骤，讲一些可以提高性能的方法。

15.2.1 提高读文本文件的性能

当从文本文件中读取数据时，有一系列操作都可能是性能瓶颈。为什么读文件的速度比你想象的要慢，可能有很多原因。为了深入调查这些原因，我们先看一下在读文件时发生了哪些事情。

首先，读取文件步骤要从磁盘中读取文件所在的数据块。如果从一个慢速磁盘读取，读取文件步骤的性能肯定也会随之变低。另外磁盘的冗余时间（寻道时间）也会影响磁盘读取性能。如果你要求“CSV文件输入”这样的步骤每次只读取比较小的数据块，那么它在每次读取下一个数据块时，都会花费一定的磁头重定向的时间。

当从文件中读取数据后，文件输入步骤会把读取到的字节转换为字符。这些字符就构成了文本或者字符串。把字节转换成字符所使用的编码必须和文件原来的编码相同。文件编码不同，就要使用不同的编码来转换。把字节转换为字符这个过程非常重要，因为字符才是用户想要获取的结果。但是，使用不同的编码方式来转换字节非常占用CPU资源。对某些双字节编码来说，如UTF-16，在文件读取步骤中几乎一半的时间都用于把字节转换为字符。

在获得字符以后，就要把这些字符分割成不同的字段。如“CSV文件输入”和“文本文件输入”步骤会根据回车换行符读取每一行文本，然后把这行文本分割成若干字段。如果是固定宽度的文件可以通过计算字符数来分割字段，否则就要通过搜索分隔符来分割字段。

最后，抽取到的字段可以被转换成几种数据类型，如Date、Number、BigNumber或Boolean。日期和数值类型的转换也比较消耗CPU。对于日期类型，要处理很多情况，例如日期格式、闰年、闰秒、时区等。对于数值类型，解析时要考虑格式、数值的分组符号、小数点符号以及指数格式等。

所有这些操作都依赖计算机的处理性能。因此，当读取带有很多字段而且要做数据类型转换的文本文件时，通常都要消耗大量的CPU时间。

除了这些基本的操作，字段里的数值可能还要做trim操作，或者当遇到空值时使用默认值来代替。

现在，我们知道了问题所在，我们再看看有哪些技巧可以解决这些问题，加快读取文件的速度。

读文本文件时使用延迟转换

字符编码和数据类型转换导致的CPU 消耗似乎无法避免。但是在一些场合下，可以使用“CSV文件输入”或“固定宽度文件输入”步骤的“延迟转换”选项。如果选中了这个选项，所有和数据相关的转换，如字符编码、数据类型转换、trim操作，等等，都将尽可能地被延迟。输入步骤只是以二进制方式读取文件，并分成若干个字段。

很明显，如果后面的步骤需要这些读取到的数据字段，数据转换操作还会进行。如果做了数据转换，还会影响转换的速度。但是，在下面一些情况下，使用延迟转换可以提高性能。

- 如果大多数字段只是简单地以同样的格式写入到另一个文本文件。
- 如果要存储数据，使用这种串行化的方式写入到磁盘会更快。
- 对于不需要转换的批量加载到数据库的情况。批量加载工具一般都是直接读取文本文件，而且生成文本的速度更快。这种方法和直接使用“表输出”步骤不同，因为“表输出”步骤往往都要进行数据格式转换。

如果在转换中只有少量的字段要转换，可以使用“字段选择”步骤的“元数据”标签里的功能来做转换。

单一文件并发读取

选中“CSV文件输入”和“固定宽度文件输入”步骤的“并行读取文件”选项，可以使用并发的方式来读取一个文件。该选项为每一份独立运行的步骤复制分配文件的一部分。步骤拷贝数通过步骤右键菜单的“改变拷贝的数量”来设置。另一种方法是在集群环境下的多个从服务器中运行这个步骤。第16章将讲述如何在集群环境下运行步骤。

说明：因为“CSV文件输入”步骤的并行读取算法以文本内的回车和换行符为切割文本的标识。所以只有在字段内容不包含回车和换行符的前提下，才能并行读取文件。

使用并行读取方法后，性能瓶颈就转换为服务器磁盘的读操作的能力。多线程读同一个文件带来的性能提高可能会因为磁头不断地重定位带来的额外负载而抵消。这种额外负载的大小则严重依赖于硬盘读操作的能力和操作系统使用的缓冲数量。

多文件并行读取

因为多线程或进程读同一个文件会产生额外负载，所以通常来说，最好使用“步骤的一个拷贝”来读一个文件。如果你有多个近似的文件，就可以让多个步骤复制读取这多个文件，如图15-10所示。

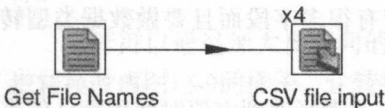


图15-10 给多个步骤复制分配多个文件

在这个例子里，转换把文件名轮流发给四个步骤拷贝。每个“CSV文件输入”步骤拷贝都会接收到不同文件名。每个步骤拷贝并行读取接收到的文件。

配置 NIO 块大小

“NIO 缓存大小”这个参数对性能也有影响，可以在“CSV文件输入”和“文本文件输入”步骤里设置这个参数。这个参数决定了从文件中一次读取的数据块大小。不会有一个明确的规则来指导你如何设置这个参数。如果这个参数设置得太大了，每次读取数据的时间会相对长一些，会降低转换的并发程度。如果你的磁盘读操作性能比较好，可以设置大一些的缓存，可以充分利用硬盘读操作的性能。若你的磁盘缓存比较大或者磁盘的寻道时间比较短，如果把缓存设置小一些，运行可能会更快。总之有很多因素都可以影响这个参数的设置，试试不同的值，看哪个运行得更好。

改变磁盘和读取文本文件

如果慢速磁盘是影响性能的根源，就要考虑把文件重新放置到一个位置。尤其是数据从远程数据库读取出来，然后被放置到临时磁盘上。而这些临时磁盘的性能往往不太好，尤其当这些临时磁盘是远程挂接到系统上时，这种情况会更严重。通过网络挂接的存储设备可以完全用于存储输入文件，但在高性能需求的场合下，需要更注意磁盘和网络的性能问题。

提高写文本文件的性能

影响读文本文件性能的一些因素，如字符编码、数据格式转换等，同样也影响写文本文件的性能。在写文本文件时，Date或Numeric等数据类型需要先转换为文本，然后文本要根据编码转换为二进制格式保存。用于提高读文本文件的性能的方法同样也可以用于提高写文本文件的性能。

15.2.2 写文本文件时使用延迟转换

对于读一个文本文件，然后再写成另一个文本文件的情况，字符编码和数据转换的开销要发生两次。在这种情况下，前面说过的延迟转换将有助于性能的提升。

并行写文件

并行方式写一个文件是不可能的。不能使用多个“文本文件输出”步骤拷贝写到同一个文件中。可以试验一下，得到的将是一个行和列的位置都混乱的文件。高级的锁算法可以解决这个问题，当然也会带来降低并发度的问题。

当然最简单的解决方案，是把数据写到多个文件中。如果需要，可以把这些文件保存在多块磁盘上，以便提高将来的扩展能力。

把数据写到多块磁盘的多个文件里，最简单的方法是使用“文件名中包含步骤数”选项或者在文件名中使用内部变量\${Internal.Step.Unique.Number}，把每个要写入的磁盘创建符号链接，如/disk1、/disk2，再在文件名中使用上面的内部变量，就可以往多块磁盘上并发写多个文件。

注意内部变量`${Internal.Step.Unique.Number}` 也可以用在集群环境下, 另外在集群环境下还有内部变量`${Internal.Step.Unique.Count}`, 步骤拷贝总数, 也可以使用。

改变磁盘和写文本文件

通常来说写磁盘的所有的时间多是读磁盘所需时间的两倍。如果你不能更换磁盘, 由于写磁盘所带来的性能下降是非常明显的, 这时可以考虑使用压缩文件。压缩文件可以明显减小输出文件的大小, 在一些情况下甚至可以压缩到原大小的十分之一。在高压缩比的情况下, 磁盘的写性能就不是问题了。

15.2.3 提高数据库性能

在典型数据仓库场景下, 你要从各种关系型和非关系型数据库中读取或写入数据。在大多数情况下, 我们都要使用数据库驱动来完成这个工作。所以在读写数据库操作时都要通过中间的一个抽象层, 我们来详细看一下读写数据库的详细过程。

如果在步骤里指定了数据库连接, 通常是创建一个数据库连接名。这样步骤就可以找到数据库的元信息。在转换运行阶段, 在步骤的初始化阶段会建立数据库连接。如果所有步骤都可以成功建立数据库连接, 转换就开始运行。此时就可以往数据库中读写数据。

需要说明的是每个步骤拷贝都会创建一个独立的数据库连接。就是说, 如果你为某个步骤创建了10个步骤拷贝, 最后就会创建10个数据库连接。

我们也可以改变这种默认设置, 选中“把转换放入数据库事务中”(使用唯一连接)选项。选中该选项后, 转换将为用到的每个数据库只创建一个连接。当转换被成功执行后再提交事务。当转换运行中有任何错误事务将回滚。

避免动态SQL

一旦建立了数据库连接, 能在前端做的工作尽量在前端做。对大多数步骤而言, 就是SQL语句的编译, 也称为语句的预处理。这是一个消耗时间的过程, 因为数据库要验证表和列是否存在、选择索引、设置连接条件, 等等。如果SQL语句对所有要处理的数据行都是一样的, 我们希望这种编译只进行一次。

“执行SQL脚本”、“执行数据行SQL脚本”、“动态SQL行”步骤里, SQL语句没有做预编译处理。因此这些步骤都会产生一些性能问题, 这些步骤比“表输入”、“表输出”、“数据库查询”等单一功能步骤的性能要低。

处理数据的往返操作

当SQL语句编译完, 来自前面步骤的数据值也和SQL语句里的参数绑定后。这些参数值被传到数据库, 数据库会执行SQL语句, 并返回结果。这就意味着对于大多数数据库操作而言, 都有一个数据往返的过程。

因此和数据库相关步骤的性能, 严重依赖于这种往返操作的速度。而这种往返操作的速度取决于网络的速度、网络的延迟、数据库的性能。

减少网络延迟

通常我们还要考虑数据库是运行在本地机器上、本地网络上或者通过VPN 访问的远程网络上。由于网络延迟不同，这三种情况下的性能差异很大。

通常使用ping 命令来判断网络延迟。表15-1显示了几种不同情况下的网络延迟以及转换的最大吞吐量。

表15-1 网络延迟的影响

| 描述 | 延迟 | 最大吞吐量 |
|-------------|---------|-----------|
| 本机 | 0.035ms | 28 571行/秒 |
| 本地网络 | 0.800ms | 1 250行/秒 |
| 服务器位于本国 | 10.5ms | 95行/秒 |
| 服务器在其他国家 | 150ms | 400行/分钟 |
| 通过卫星线路的移动网络 | 1500ms | 40行/分钟 |

对于网络延迟，我们通常无能为力。因此只能研究是否有其他可能的解决方法。

第一个解决方法是看能否减少数据库操作的往返次数。例如，把要查询的数据加载到内存里（一次数据库往返可以获得所有行），或者使用缓存（对每个唯一键值有一次数据库往返）。用于数据库查询操作的步骤通常都有一个数据缓存选项，流查询步骤可以把所有的数据行加载到内存，所以流查询步骤的查询速度非常快。也可以使用“结果集连接”步骤以数据流的方式连接两个大的数据集，这样内存消耗小。“结果集连接”步骤要求输入的数据事先排好序。但通常使用数据库来做排序操作，因为数据库排序要比基于流的引擎排序更快（见下面的章节），尤其是排序字段已事先建立索引。

第二个解决方法是同时执行多个步骤拷贝。这操作起来非常容易。理论上，如果有两个步骤拷贝，延迟时间就会缩减为原来的一半。但在实际中，因为数据库处理连接请求的方式不同，或者网络饱和等原因，性能往往能增长30%~50%。

最后，批量处理也会让性能有很大不同。例如“表输出”步骤里的“使用批量更新”选项通过把数据行放到大的批量提交数据块里，可以减少数据库往返操作的次数。在网络比较快的情况下，使用多个步骤拷贝可以极大提高转换性能。在吞吐量上可以提高20%~40%。

网络速度

很明显，影响转换性能的不只是网络的延迟。网络的速度，也就是带宽，也会影响到性能。当有很多的数据需要通过网络来处理，网络的带宽就更重要了。如果要使用“表输出”或“插入更新”步骤加载数据，要通过网络和数据库交换大量的数据。

正如前面所说，你可能对网络的性能问题无能为力。但是，可以在加载数据之前，把数据移动到和数据库尽可能近的地方。我们此时可以想到行业里的一句谚语“一辆装满了硬盘在高速公路上奔驰的大货车的带宽是无可比拟的”。换句话说，对于一些慢速网络，可以使用一些非常规的手段，如把数据放到一块硬盘上，使用隔夜快递，把它快递到世界的另一端。

说明：在加载之前把数据移动到数据库附近的一个例子就是批量加载。这一方法通常用于把大量数据直接从文本文件中加载到数据库里，Kettle 支持很多种数据库的批量加载。

处理关系型数据库

前面说过，数据库本身的性能对转换的性能也起着重要作用。而数据库调优需要很高的技巧。下面有一些通用的调优规则可以应用于各种关系型数据库。

批量提交大小

在支持批量提交的步骤里，根据使用的数据库的不同，可以尽量将批量提交数设置得更大一些。例如在“表输出”步骤里，可以设置“提交大小”参数，这样关系型数据库就会使用事务。一般来说，这个值设置得越大，性能会越好。但任何好的事情过度了，就会向坏的方向发展，所以要调整一个适当的值，使性能最好。一般在普通机器上如果速度到了1~2万行每秒，再调整这个参数，可能就很难再提高性能了。

索引

在做查询或连接等操作时，表中是否创建了适当的索引也至关重要。例如，图15-11 显示了在没有索引情况下的“维度查询/更新”步骤的性能图。

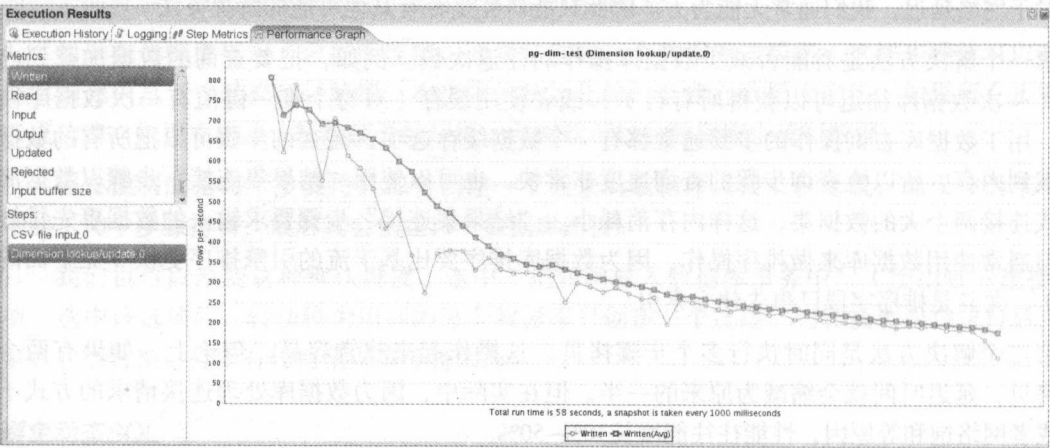


图15-11 没有索引情况下的性能图

正如你看到的，尽管数据库缓存了尽可能多的数据。数据库的性能还是在持续下降，因为数据库在做一种所谓的全表扫描的操作，来找到数据行的位置。如果数据库里有N行，为找到其中一行，平均要比较N/2次。因为表里的记录在不断增多，N/2这个数字也在不断变大。所以缓存实际并没有起到太大作用。如果没有索引，比较次数会持续增长，直接的结果就是查询会越来越慢。而使用了索引的查询就不严重依赖于要查询的行数。图15-2 显示了在有适当索引的情况下完成相同工作时的性能图。可以看到，索引创建后，性能会相对稳定。

同时，你也要知道索引会使数据库的更新、插入、删除操作变慢。在关系型数据库里创建的索引越多，这些操作就会变得越慢。因为这些原因，在做大规模操作时，如数据初始化、批量更新等操作，可以先删除索引。在做完这些大规模操作后再创建索引，会更节省时间。

表分区

表分区用来把一个大表拆分成若干个小表，分区规则通常很简单，如按照一定范围等。例如，一个表分区可以包括一天的数据、一个月的数据或者一年的数据。另一种分区规则是按照简单的哈希值。例如，把所有客户名字里首字母是A的客户放在一个分区表里。

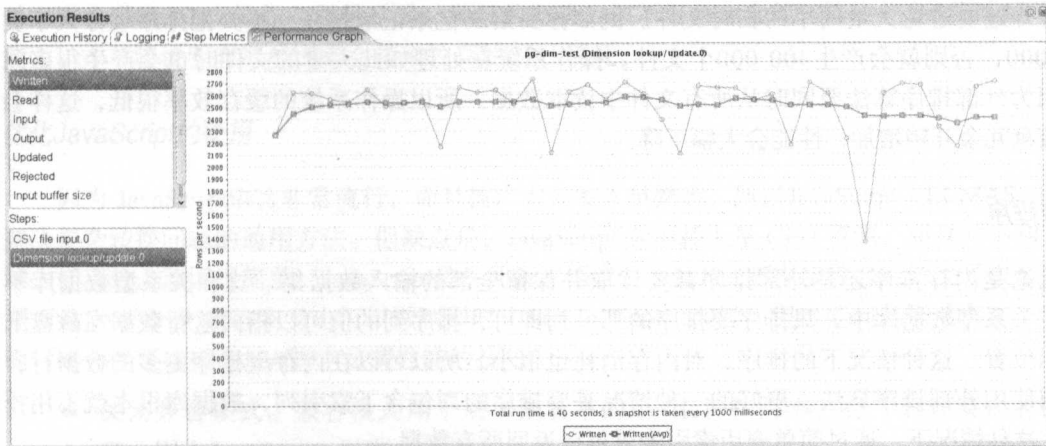


图15-12 适当的索引使性能保持稳定

无论分区是按照什么逻辑来划分的，都应该要让数据库能快速知道一条记录是在哪个分区里。这样在使用索引之前，就缩小了查询范围，相应就会提高查询性能。

分区的另一个优点就是，它可以提高加载和查询数据的并行度。一些关系型数据库允许以并发方式向分区里加载数据，这样会提高吞吐量，提高整体性能。

约束

数据库里的约束会降低数据库操作的性能。因为每一次的插入、更新、删除操作都要验证约束条件。因此，通常在做大规模数据操作之前都要禁止约束。在多维数据仓库的开发实践中，通常是在开发环节设置约束条件的检查。因为如果可以通过ETL过程中来查询外键，就没有必要在数据库里再建立外键等约束。

触发器

另一个要考虑影响性能的问题是数据库表里的一个或多个触发器。触发器实际也是数据库的存储过程，当对数据库表有插入、更新、删除等操作时才被执行。根据配置的不同，触发器会严重影响对数据库大规模操作时的性能。在实践中，数据仓库的表里通常都不会有触发器。

15.2.4 数据排序

因为 Kettle 转换的流特性，数据排序也是一个有趣的性能问题。基于流的排序的主要问题是你不知道要排序的最大行数。如果你假设所有要排序的数据行都可以放到内存里，那么排序当然会很快。例如要排序一百万行数据，这些数据都可以放在内存里，在这种情况下，排序只要10秒钟的时间。

但是，如果内存里只能容纳一半的数据来排序，那么就要使用所谓的外部排序。就是说先把前五十万行数据排序，放在硬盘的临时文件中。再把后五十万行数据排序，然后也放在硬盘的临时文件中，最后再从硬盘上逐行读取这些文件里的数据。

显然，由于文件串行化的性能消耗，外部排序要比内存排序慢很多。但是因为内部排序会存在内存溢出的问题，外部排序是唯一的办法。

要解决排序的性能问题，第一个要做的事情是给Kettle 转换引擎更多的内存。可以在排序对话框的“排序缓存大小（内存里存放的记录数）”输入框里设置一个比较大的数值。

另外还要避免大量的小的临时文件。例如你要给20亿条记录排序，不要把排序缓存大小设置为5000，否则就会产生400 000个文件。操作系统在处理如此大量的文件时也会产生很多问题。因为外部排序算法要同时从所有文件中读取数据，所以操作系统的缓存效率很低，这样会导致磁盘冗余开销增加，性能会大幅下降。

数据库排序

无论是内存排序还是外部排序都要读取并保留全部的输入数据集。这和关系型数据库不同。在关系型数据库中，只排序要排序的那一列即可，排序列的值可以指向这行数据在磁盘上的实际位置。这种情况下的排序，对内存消耗也很小，所以可以在内存里排序更多的数据行，而不用使用外部排序算法。更好的一种情况是要排序的列包含了索引列，数据库根本就不用排序。在这种情况下，通过简单遍历索引表就可以返回所有数据。

因为上述原因，无论何时，只要数据在数据库里，你都要尽可能使用数据库排序。
注意，如果是先把数据加载到数据库后再排序，并不一定快。如果数据并不在数据库里，那还是用Kettle来完成排序。

并行排序

正如上面描述的，排序需要计算机的处理和磁盘的I/O能力。所以可以考虑以并行的方式进行排序。可以在ETL里把数据分成几个部分，如图15-13的转换。

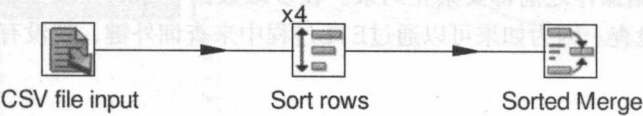


图15-13 并行排序

在这个简单的例子里，四个排序步骤的拷贝同时工作。从“CSV文件输入”步骤里读取到的数据行，轮流被分配给这四个排序步骤的拷贝。每个排序步骤的拷贝都把输入的数据行排序，然后生成四份排好序的数据集。最后使用了“Sorted Merge”步骤，把这四个排好序的数据集再合并，生成最后的排序结果。

除了使用步骤拷贝的方式，我们还可以使用集群，如图15-14所示。

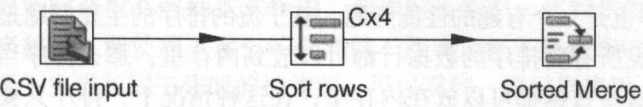


图15-14 集群排序

在集群排序的情况下，数据行被发送到四台不同的服务器上。很容易想象到，通过这种集群可以获得四倍的处理能力，同时也需要四倍的磁盘 I/O 吞吐。如果数据通过网络往返带来的额外延迟少于集群排序带来的时间收益，那么集群方式排序就是有意义的。通过在Amazon EC2这样的云计算平台上的测试表明，并行方式排序可以带来很大的性能增益。

15.2.5 减少CPU消耗

转换以多线程的方式把数据流从一个步骤发送到另一个步骤。除非绝对需要，转换引擎将

尽力避免磁盘的I/O 操作。尽管这是一种好的做法，但不可避免会使转换性能受制于CPU的处理能力。本节将说明有哪些因素会带来较高的CPU 消耗。

优化JavaScript的使用

因为 JavaScript语言非常流行，而且被广大开发人员熟悉，所以JavaScript（ECMAScript）是解决复杂转换问题的通用方法。但缺点是，JavaScript 会消耗大量 CPU 资源，而且并不和其他步骤一样高效。原因很简单：JavaScript是一种脚本语言。尽管Java脚本步骤已经解决了很多性能问题而且JavaScript 语言本身也可以做到即时（Just In Time）编译。但性能问题仍然存在。所以下面讨论一些可以提高Java脚本步骤性能的技巧。

- **关闭兼容模式。**兼容模式用于从早期版本的Kettle移植脚本。打开兼容模式的代价很高，因为JavaScript 引擎会使用一个兼容层来判断脚本里的每一行。关于如何把Kettle 2.x里的JavaScript 移植到结构更清楚的Kettle 3.0的JavaScript，请看<http://wiki.pentaho.com/display/EAI/Migration+JavaScript+from+2.5.x+to+3.0.0>。
- **避免JavaScript。**过去很多需要JavaScript来完成的功能都不再需要JavaScript。出现了很多新的步骤来实现以前的这些功能，而且更便于维护和提高性能。
- **数据转换。**可以使用“字段选择”步骤代替Java脚本步骤，字段选择步骤里的元数据标签可以把数据从一种类型转换到另一种类型。
- **创建字段拷贝。**“字段选择”步骤也会做字段的拷贝。或者使用“计算器”步骤的“创建字段A的拷贝”的功能。
- **从上一行获取数据。**可以使用“前后行查询”步骤来解决这个问题。这个步骤可以查询前或后某行的数据。关于该步骤更多介绍请看第10章。
- **将一个字段拆分为多行。**使用“列拆分为多行”步骤可以把一列一行数据拆分为一列多行。
- **数值范围。**Java脚本步骤里使用if-then 结构，不便维护而且容易出错，可以使用“数值范围”步骤代替。
- **随机数和GUID。**“生成随机数”步骤可以快速生成各种类型的随机数，也可以用于Globally Unique Identifiers（GUID）。
- **校验。**“增加校验列”步骤可以给一行数据增加各种类型的校验值。
- **写一个步骤插件。**如果你发现你要一遍遍使用JavaScript来解决同一个问题，那么你最好自己写一个插件。插件可以把JavaScript 要实现的功能封装起来，同时提供一个用户界面，还可以提高性能。唯一不便的地方在于需要开发人员有Java 开发经验。另外还要单独建一个和Kettle 无关的插件工程，关于插件开发的更多内容请看第23章。
- **用户自定义Java类步骤。**该步骤可以让你直接在步骤的对话框里写Java 代码，用来实现类似插件的功能。这个代码在运行时编译，也有比较好的性能。关于这个步骤请见第23章。
- **合并Java脚本步骤。**如果你还是觉得必须要写JavaScript，应尽量避免多个独立的Java脚本步骤，把多个Java脚本步骤合并成一个步骤。因为把数据流里的字段值作为Java脚本步骤上下文的变量也会有性能开销。
- **创建变量。**如果Java脚本步骤里定义的变量只需要在转换开始时定义一次，那么最好把这些变量定义语句放在一个独立的脚本里，并在Java脚本步骤对话框里用右键单击脚本

的标签名, 把这个脚本设置为“启动脚本”。创建一个JavaScript对象非常容易, 但要消耗CPU 周期, 所以不要给转换里的每一行都分配一个JavaScript对象。

- **添加静态值。**可以使用例如“增加常量”或“获得变量”来设置静态值。这些步骤性能较好, 且易于维护。

启动一个步骤的多份拷贝

如果你有一个步骤可能会消耗大量CPU 资源, 例如Java脚本步骤或正则表达式步骤, 你就要考虑使用多份步骤拷贝。因为现在的计算机系统都有多个内核, 就要充分利用这些内核。使用多份拷贝启动一个步骤可以把负载分配给这些内核, 从而提高性能。对于一个比较慢的步骤, 使用多份拷贝将是最后的一个办法。步骤本身需要先优化。确保优化JavaScript代码或正则表达式, 这些才是引起性能问题的根本原因。

并行运行的另一种情况是分区。在Kettle 里, 步骤的每一个分区都由步骤的一份拷贝来处理。就是说分区键值一样的记录都被发送到了一个步骤拷贝里。这样一些不容易并发的工作也可以并发执行, 如内存分组操作。看如图15-15所示的例子。

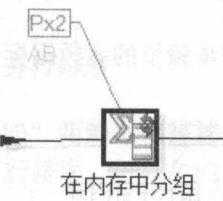


图15-15 把分组步骤进行分区, 把负载分散到多个处理器上

在这个例子里, 我们按照分组键进行分区。这样保证每个步骤拷贝的分组结果是正确的。同样的原理也可以应用到缓存策略, 分区可以使每个步骤拷贝不必保存所有数据, 只要同一个数据行都能从同一个步骤拷贝中找到。缓存的命中率就会提高, 从而会提高性能。

选择和移去字段

“字段选择”步骤一般用来选择和移去字段。但通常这么做会加重CPU的负载。这两种操作都会强迫系统重新创建每个输入数据行。然后这些新的数据行再按照正确的顺序被赋予新的数据值。

在Kettle的早期版本中, 你不得不通过“字段选择”步骤来设置要输出的数据。类似“表输出”这样的步骤都不支持字段选择。在最近版本的Kettle, 在表输出时可以用不用“字段选择”步骤。

还有一种要使用字段选择步骤的情况就是从多个步骤把数据发送到同一个步骤, 这时要求每个步骤发送数据的格式(字段个数、字段名、字段类型)相同。这时可能就要用字段选择调整数据格式, 但也可以考虑使用“增加常量”步骤使输入数据流的字段格式相同。“增加常量”步骤的速度比“字段选择”步骤的速度要快。

在一些场合下, 如把所有的数据行放在内存里或把数据行回写到硬盘等, 选择尽可能少的字段将对性能和内存消耗都有益处。例如, 对数据排序时, 数据库查询缓存数据时, 或把数据通过集群传递到子服务器时。

管理线程优先级

从 Kettle 3 开始, 在转换设置里多了一个“管理线程优先级”的选项, 选中该选项可以管理步骤线程。该选项用来解决 Kettle 运行所依赖的 Java 虚拟机在多线程处理方面的一些限制。当步骤之间的行集处于满的状态或者空的状态的时候, 步骤线程仍旧会频繁地给行集加锁。打开该选项后, 加锁步骤的优先级会下降, 尽量避免线程争用情况的发生。

如果是从早期的 Kettle 版本移植过来的转换, 或者不小心没有选中这个选项, 你都可以试试打开这个选项。在大多数情况下, 并非所有情况, 该选项会对性能的提升有帮助。

给数据行添加静态数据

如果你使用多个步骤只生成一个数据, 那么最好不要把这些步骤放在数据的主流程里。例如图 15-16 的转换。

在这个转换里, 前三个步骤先获得两个变量放在两个字段里, “计算器”步骤把这另一个字段连接起来, 生成一个新的字段, “字段选择”步骤再做类型转换, 把生成的字段转换为 Date 类型字段。对于整个数据流来说, 所有输入行的最后都会增加一个一样的 Date 类型的数据。每个输入行都会重复上面的计算过程。实际上, 只需要计算一次即可。可以使用“笛卡儿乘积”步骤来实现这个功能。这样可以把转换调整成图 15-17 的样子。

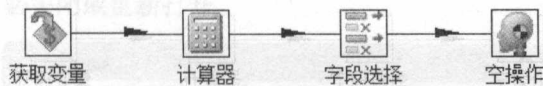


图15-16 多次计算一个常量值

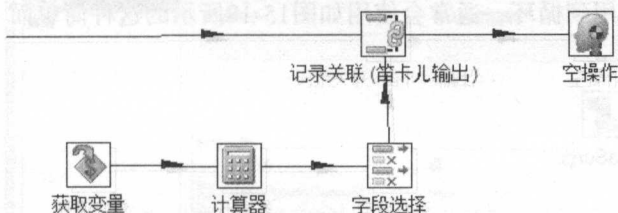


图15-17 只计算一次常量值

注意要在“笛卡儿乘积”步骤里正确设置“主数据流”选项。这样从“字段选择”步骤过来的一个数据会保持在内存里, 而不是主数据流里的数据被保存在内存里。“笛卡儿乘积”步骤配置完成后, 例子里的功能就很容易实现了。

限制步骤拷贝的数量

转换在运行时会为每个步骤创建一个线程。每个线程是并发执行的任务。因为计算机在一个时刻只能执行一条指令, 并发或多线程实际是计算机处理器模拟的。这种模拟是通过任务之间互相切换来完成的, 每个任务一次执行时只占用很小的时间片段。这个规则也有特例, 就是线程可以运行在不同的处理器上, 此时就不必再做任务的切换。

多任务或并行处理的直接结果就是任务的切换会带来性能的开销, 因为处理器在切换到另一个任务时, 需要记住当前在执行任务的状态, 在切换回来后还要把这些状态再恢复回来。

尽管任务切换的负载并不大，但如果在一个系统上运行很多个线程，这种负载会累加。因为在转换里每个步骤拷贝都是独立的线程，所以你会发现如果转换里的步骤数量太多，转换的性能会下降。因为不同计算机系统之间，线程之间切换的效率是不同的，所以如何设计转换很难有明确的规则。但是如果性能问题比较重要，作为一般性的规则，在设计转换时，要注意转换里步骤的数量应该少于服务器内核总数的三到四倍。

避免过多的日志

你在调试问题时，在运行转换时把日志设置为较高的级别，如调试（Debug）级别，这样可以记录更多的内容。但要记住：日志记录的内容越多，CPU和内存的开销就会越大。所以不建议在运行转换或作业时把日志级别设置为大于详细（Detail）的级别。如果想在每天晚上运行作业时，记录更多的日志，要确保转换需要的时间不能超过规定的时间窗口。关于日志更详细的内容请参考第14章。

15.3 提高作业性能

一般遇到的性能问题大部分都是转换的性能。但在设计作业时也有一些性能问题要注意。本节主要讲一下如何提高作业里循环的性能，如何提高作业里短时间运行的、有数据库连接的转换的性能。

15.3.1 作业里的循环

很多Kettle的初学者，如果要在作业里用到循环，通常会使用如图15-18所示的这种简单而直接的方式。

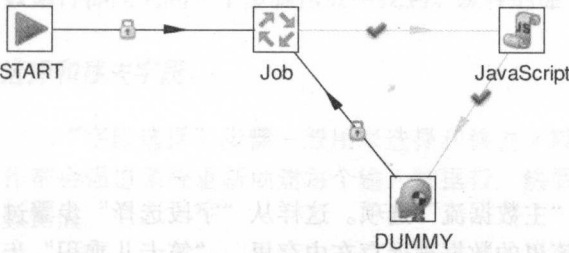


图15-18 一种简单但是低效的作业循环

这个例子的场景是，作业里的子作业项每次处理一个文件，JavaScript作业项判断是否还有其他文件要处理，如果还有文件就继续循环。

如果只需要进行几次循环，可能不会有任何问题。但是，如果循环的次数非常多，你就会发现这种方法非常慢。有性能问题的主要原因是作业里有两个额外的作业项，以及反复加载作业项的元数据。这些额外的工作会使作业变慢。除了作业会变慢，由于作业使用了回溯算法，还存在内存堆栈溢出的风险。算法参考第2章。

再看另一个更高效的作业，如图15-19所示的作业里有一个转换作业项，作业项名字是“Get filename”，该转换作业项可以获取一组要处理的文件名。该转换里的最后一个步骤是“复制行到结果”步骤，该步骤把获取到的文件名列表传递到上级作业里的下一个作业项。



图15-19 在作业里做循环的正确做法

注意还要在作业作业项的设置对话框里选中“对每个输入行执行一次”选项，这样作业作业项会循环结果里的每一行。这种方法的优点是不会占用太多的堆空间，而且作业元数据只被加载一次，作业项只被执行一次，所以性能会有很大提高。

15.3.2 数据库连接池

对于需要反复读取数据库以获取少量数据的作业，例如前面例子中的作业循环，你会发现反复建立连接和断开连接会引起很大的性能问题。Oracle和一些集群数据库在建立连接时非常慢。如果要获取的数据非常大，获取这些数据要一些时间，那么花一些时间建立连接还是可以接受的。但是，如果有大量的小文件要单独处理，数据库连接延迟本身就会限制作业的性能。

如果你遇到了连接过慢的问题，可以试一下“数据库连接”对话框里的“使用连接池”选项，如图15-20所示。

Kettle 里的数据库连接池使用了Apache DBCP 项目 (<http://commons.apache.org/dbcp/>)。它实际是创建了多个开放的数据库连接，对于使用数据库连接的转换或作业，这些数据库连接不必关闭或重新打开。

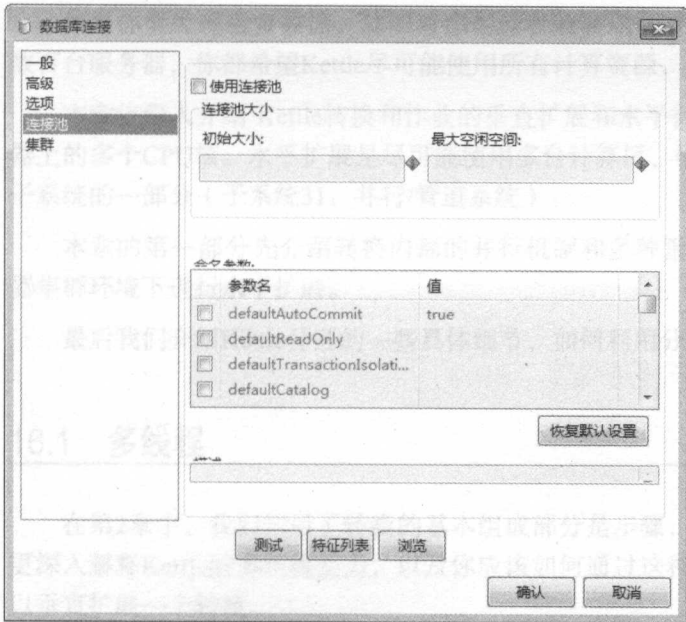


图15-20 使用数据库连接池

15.4 小结

本章我们了解了转换引擎如何工作，如何通过简化的方法和查看度量值的方法来检查转换的性能瓶颈。

第16章 并行、集群和分区

如果你要处理很多数据，就要考虑如何有效使用所有的计算资源。不管是个人电脑，还是数百台服务器，你都希望Kettle尽可能使用所有计算资源，并在一定时间内获取执行结果。

本章将深入介绍 Kettle转换和作业的垂直扩展和水平扩展。垂直扩展是尽可能使用单台服务器上的多个CPU核。水平扩展是尽可能使用多台计算机，使它们并行计算。这两种方法都是ETL子系统的一部分（子系统31，并行/管道系统）。

本章的第一部分先介绍转换内部的并行机制和多种垂直扩展方法。然后介绍怎样在子服务器集群环境下进行水平扩展。

最后我们介绍Kettle分区的一些具体细节，如何利用分区进一步提高并行计算的性能。

16.1 多线程

在第2章中，我们知道了转换的基本组成部分是步骤，而且每个步骤是并行执行的。现在将更深入解释Kettle的多线程能力，以及你应该如何通过这种能力，来使用计算机所有的计算资源以垂直扩展一个转换。

默认情况下，转换中的每一个步骤都在一个隔离的线程里并行执行。但我们可以为任何一个步骤增加线程数，我们也称之为“拷贝”。在15章里我们也解释过，这种办法能够提高那些消耗大量CPU时间的转换步骤的性能。

看一个简单的例子，如图16-1所示，使用一个“User Defined Java Class”步骤来处理所有数据。

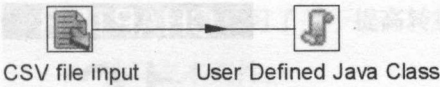


图16-1 一个简单的转换

右键单击这个“User Defined Java Class”步骤，选择菜单中的“改变开始复制的数量”，如果你指定了四份，将看到转换如图16-2所示。

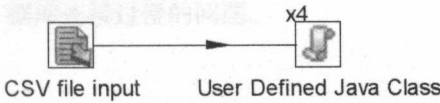


图16-2 在多个拷贝下运行一个步骤

这个“x4”的符号说明这个步骤有四份拷贝同时运行。

注意所有步骤拷贝只维护一份步骤的描述，为了方便理解下面的章节，这里定义几个专业术语。

- step: 描述要做的工作的定义或元数据。
- step copy: 执行某项工作的一个并行工作线程。

也就是说，一个step仅是一个任务的定义，而一个step copy则表示一个实际执行的任务。

16.1.1 数据行分发

在这个例子里，我们把一个步骤拷贝里的记录发送给四个步骤拷贝，这些记录是怎样分发给目标步骤拷贝的呢？默认情况下，分发工作使用轮询方式执行。也就是说如果有N份拷贝，第一份拷贝获取第一条记录，第二份拷贝获取第二条记录，第N份拷贝获取第N条记录。记录N+1又分发给第一份拷贝，依此类推，直到没有记录分发为止。

这里还有另外一个比较少用的功能，使用它可以将所有记录发送给所有步骤拷贝，可以在步骤的上下文菜单中启用这个“复制发送模式”选项。这个选项会把记录发送给多个步骤，例如，同时往数据库表和文件里写入数据。在例子中，会出现一个警告对话框，如图16-3所示，询问你要选择哪个选项。

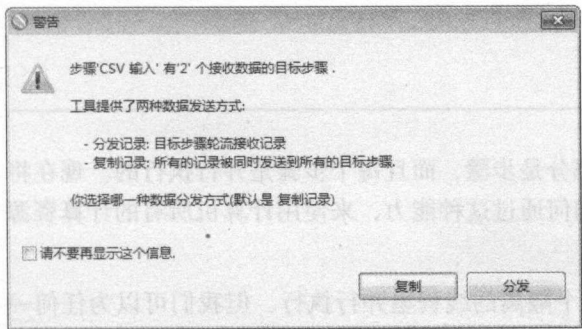


图16-3 一个警告对话框

因为要把记录复制给下面所有的步骤，所以单击“复制”按钮，转换结果如图16-4所示。

通常情况下每一条记录仅仅处理一次（而不是处理多次），所以复制的情况使用的比较少。在余下的章节中使用的都是分发的例子，而不是复制的例子。

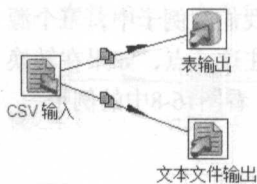


图16-4 复制数据到多个目标步骤

16.1.2 记录行合并

当有几个步骤或者一个步骤的多份拷贝同时发送数据给单个步骤拷贝时，就会发生记录行合并。图16-5显示了两个例子。

从“文本文件输出”和“添加序列”步骤来看，并不是依次从每个数据源逐条读取数据行。如果这样做会导致比较严重的性能问题，例如一个步骤输送数据很慢，而另一个步骤输送数据很快。实际上数据行都是从前面步骤批量读取的。

警告： 不能保证从前面步骤的多个拷贝中读取记录的顺序！

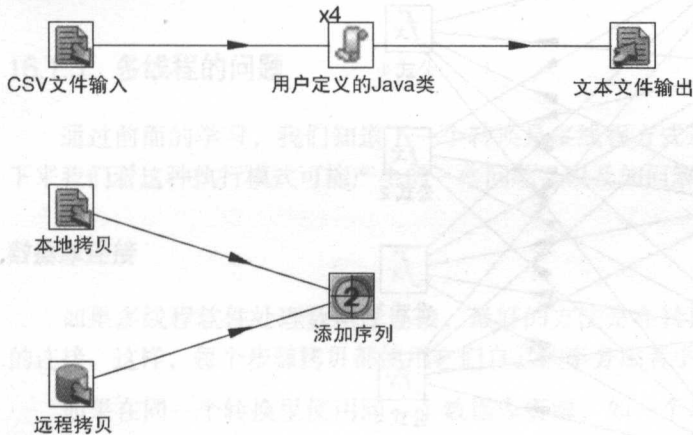


图16-5 合并记录行

16.1.3 记录行再分发

记录行再分发是指：X个步骤拷贝把记录行发送给Y个步骤拷贝，如图16-6所示。

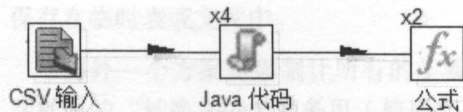


图16-6 记录行再分发

在这个例子中，每个User Defined Java Class步骤拷贝都把记录行分发给两个目标步骤拷贝。这个结果等同于图16-7的转换。

这个再分发算法的主要优势是把记录行平均分配给每个步骤拷贝，防止某个步骤拷贝有很多工作，而其他步骤拷贝只有很少工作。

从图16-7可以看出，在UDJC和Formula步骤之间有 $X*Y$ 个行缓冲区。在我们的例子中，三个源步骤和两个目标步骤之间有六个缓冲区（箭头）。在设计转换的时候要记住这一点，如果在转换末端有很慢的步骤，这些缓存都可能被填满，这样会增加内存消耗。例如，看图16-8中的例子。

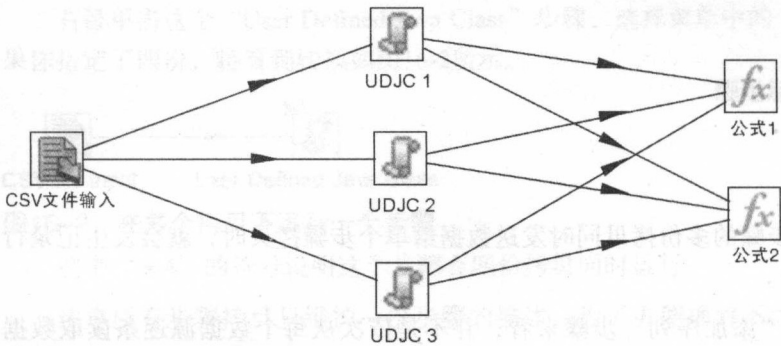


图16-7 记录行再分发展开

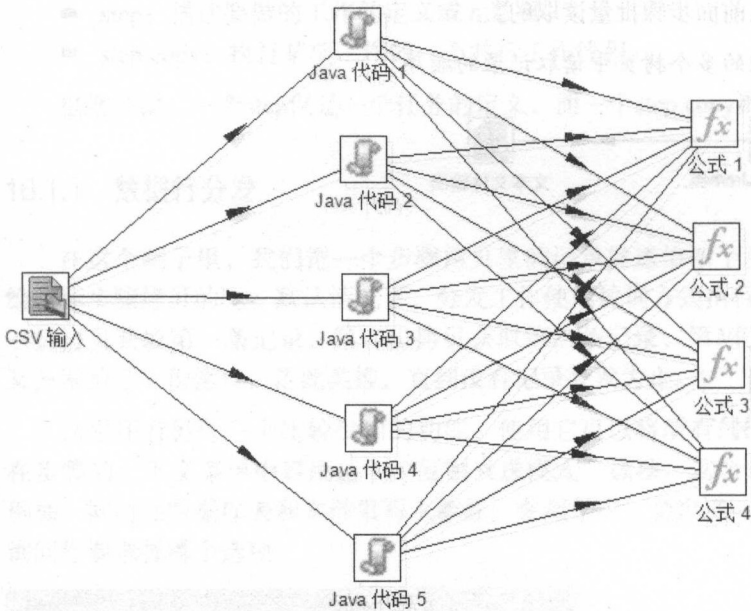


图16-8 再分发会分配很多缓冲区

尽管你在转换图里只看到了一条线，实际上有五个源步骤和四个目标步骤，共分配了20个缓冲区。默认情况下最大缓冲区的记录行数是10 000，所以内存中能保存的记录行总数是200 000。

16.1.4 数据流水线

数据流水线是再分发的一种特例，在数据流水线里源步骤和目标步骤的拷贝数相等 ($X=Y$)。此时，前面步骤拷贝的记录行不是分发到下面所有的步骤拷贝。实际上，由源步骤的拷贝1产生的记录行被发送到具有相同编号的目标步骤拷贝，图16-9是这种转换的例子。



图16-9 数据流水线

在技术上，它等同于图16-10的转换。

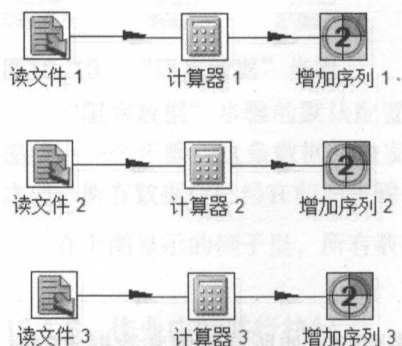


图16-10 数据流水线展开

分发和合并记录行的过程会产生一点性能开销，通常情况下，最好让连续步骤的拷贝数相等，这样可以减少开销。

这种减少步骤拷贝之间开销的过程，也可形象地比喻为将数据放进游泳池的泳道（彼此之间不受干扰）。

16.1.5 多线程的问题

通过前面的学习，我们知道了一个转换是多线程方式运行的，所有的步骤都并行运行。接下来我们看这种执行模式可能产生的一些问题，以及如何解决这些问题。

数据库连接

如果多线程软件处理数据库连接，最好的方法是在转换执行的过程中为每个线程创建单一的连接。这样，每个步骤拷贝都使用它们自己的事务或者事务集。

如果在同一个转换里使用同一个数据库资源，如一个表或一个视图，还容易产生条件竞争问题，这是一个潜在的后果。

一个常见的错误场景，就是你在前面的步骤里向一个关系型数据表里写入数据，在随后的步骤里再读取这些数据。因为这两个步骤使用不同的数据库连接，而且是不同的事务，你不能确保被第一个步骤写入的数据，对其他正在执行读操作的步骤可见。

一个常见且简单的解决这个问题的方案就是将这个转换分成两个不同的转换，然后将数据保存在临时表或文件中。

另外一个方案是强制让所有的步骤使用单一数据库连接（仅一个事务），启用转换设置对话框中的“转换在一个事务里（使用唯一连接）”选项即可，见图16-11。

这个选项意味着Kettle里用到的每个命名数据库都使用一个连接，直到转换执行完后才提交事务或者回滚。也就是说在执行过程中完全没有错误才提交，有任何错误就回滚。请注意，如果错误被错误处理步骤处理了，事务就不会回滚。

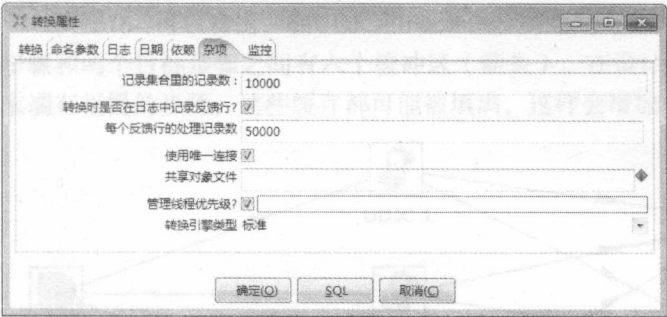


图16-11 设置一个转换的事务

使用这个选项的缺点就是会降低转换的性能，这里有很多原因，如所有步骤和数据库的通信都由一个同步的数据库连接来完成，在服务器端也只有一个服务进程来处理请求。

执行的顺序

由于所有步骤并行执行，所以转换中的步骤没有特定的执行顺序，但是数据集成过程中仍然有些工作需要按某种顺序执行。在大多数情况下，通过创建一个作业来解决这个问题，使任务可以按特定的顺序执行。

在Kettle的转换中，也有些步骤强制按某种顺序执行，下面有几个技巧。

“执行SQL脚本”步骤

如果想在转换中，在其他步骤开始前，先执行一段SQL脚本，可以使用“执行SQL脚本”步骤。在正常模式下，这个步骤将在转换的初始化阶段执行SQL，就是说它优先于其他步骤执行。

但是如果你选中了这个步骤里的“单独执行每一行”选项，那么这个步骤不会提前执行，而是按照在转换中的顺序执行，如图16-12所示。

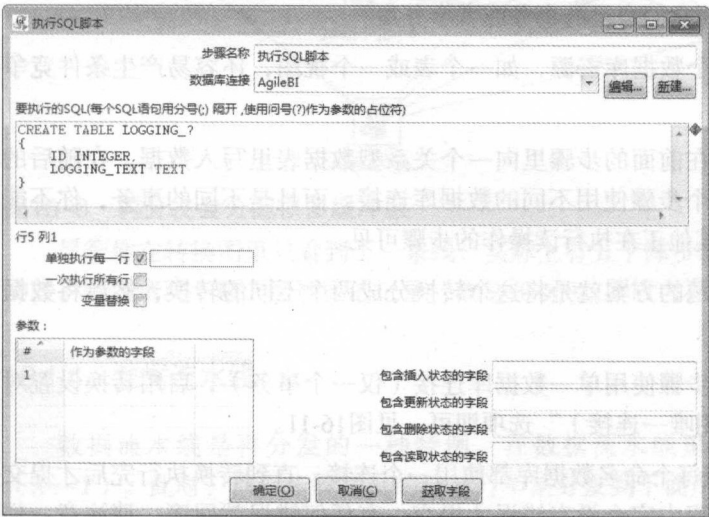


图16-12 “执行SQL脚本”对话框

“阻塞数据”步骤

如果希望所有的数据行都到达某个步骤后，才开始执行一个操作，就可以使用“阻塞数据”步骤，如图16-13所示。



图16-13 “阻塞数据”步骤

“阻塞数据”步骤的默认配置是丢弃最后一行以外的所有数据行。然后把最后一行数据传递给下一个步骤。这条数据将触发后面的步骤执行某个操作，这样你就能确保在后面步骤处理之前，所有数据行已经在前面步骤处理完。

在上图显示的例子中，所有数据行写入到数据表后，SQL语句开始执行。

16.1.6 作业中的并行执行

默认情况下，作业中的作业项按顺序执行。必须等待一个作业项执行完成后才开始执行下一个。然而，在第2章中提到过，在作业里也可以并行执行作业项。在并行执行的情况下，一个作业项之后的多个作业项同时执行，由不同的线程启动每个并行执行的作业项。

例如，你想并行更新多张维度表，可以按照图16-14的方式设计。

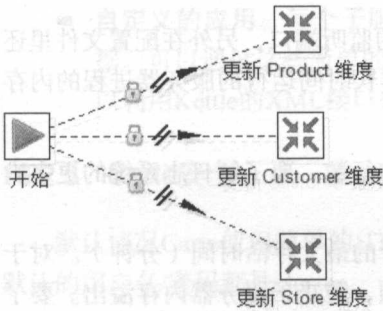


图16-14 并行更新多张维度表

16.2 使用Carte子服务器

子服务器是Kettle的组成模块，用来远程执行转换和作业。Carte是一个轻量级的服务进程，可以支持远程监控，并为转换提供集群的能力，集群将在本章的后面介绍。

子服务器是集群的最小组成模块。子服务器也是一个小型的HTTP服务器，用来接收远程客户端的命令，这些命令用于作业和转换的部署、管理和监控。

正如第3章所述，Carte程序可用于子服务器，也可用于远程执行转换和作业。

启动子服务器需要指定主机名或者IP地址，以及Web服务的端口号。例如，下面的命令将在服务器server1的端口8181上启动子服务器：

```
sh carte.sh server1 8181
```

16.2.1 配置文件

在Kettle的早期版本里，通过命令行指定配置选项。随着配置选项数目的增加，Kettle最近的版本使用XML格式的配置文件，如果你有配置文件，就可以像下面这样启动子服务器：

```
sh carte.sh slave-simple.xml
```

例子里的配置文件slave-simple.xml描述了一台子服务器的所有属性，下面就是这个配置文件：

```
<slave_config>
  <!--
    - A simple slave server configuration
  -->

  <max_log_lines>0</max_log_lines>
  <max_log_timeout_minutes>0</max_log_timeout_minutes>
  <object_timeout_minutes>5</object_timeout_minutes>

  <slaveserver>
    <name>server1</name>
    <hostname>server1</hostname>
    <port>8181</port>
  </slaveserver>
</slave_config>
```

<slaveserver>节点里描述了子服务器的主机名和子服务器的监听端口，另外在配置文件里还可以配置子服务器的其他属性。这些属性可以优化像Carte这样长时间运行的服务器进程的内存使用。

- max_log_lines：设置日志系统保存在内存中的最大日志行数。要了解日志系统的更多内容请参考第14章。
- max_log_timeout_minutes：设置保存在内存中的日志行的最大存活时间（分钟）。对于运行时间很长的转换和作业，这是一个尤其重要的选项，防止子服务器内存溢出。要了解更多内容请参考18章。
- object_timeout_minutes：默认情况下，在子服务器的状态报告中，可以看到所有转换和作业，这个参数可以自动地从状态报告列表中清除老的作业。

16.2.2 定义子服务器

在Spoon左侧的视图部分。右键单击“子服务器”树节点，选择“新建”，然后在弹出的新建对话框中，填入子服务器的具体属性，如图16-15所示。

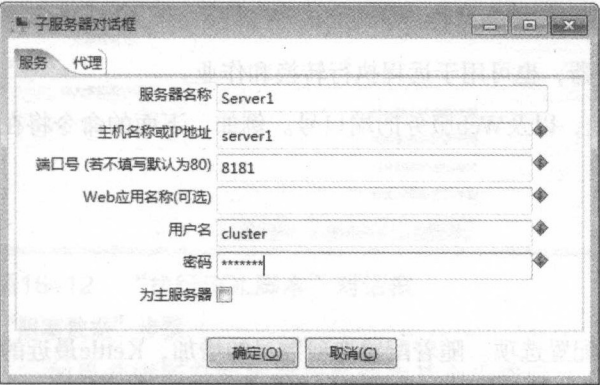


图16-15 定义子服务器

16.2.3 远程执行

在Spoon的“执行转换”对话框里可以指定要运行转换和作业的远程子服务器。如果作业或转换被另一个作业调用，可以在作业或转换作业项的对话框里选择一个远程子服务器，此时作业或者转换作业项就可以远程执行了。

16.2.4 监视子服务器

有几种方法可以远程监视子服务器。

- **Spoon**: 在Spoon树形菜单中右键单击子服务器，选择“监控”选项。就会在Spoon中出现一个监控界面，包含了所有运行在子服务器上的转换和作业的列表。
- **Web浏览器**: 打开一个浏览器窗口，输入子服务器的地址，例如http://server1:8181/，浏览器将显示一个子服务器菜单，通过这些菜单项，可以控制和监控子服务器。
- **PDI企业控制台**: 这是PDI企业版的一部分，企业控制台提供了监控和控制子服务器的功能。
- **自定义的应用**: 每个子服务器都以URL的方式提供服务，返回的结果是XML格式的数据。可以通过这些简单的Web服务与子服务器通信。如果使用了Kettle的Java库，你还可以利用Kettle的XML接口来解析这些XML。

16.2.5 Carte安全

默认情况Carte使用简单的HTTP认证，在文件pwd/kettle.pwd中定义了用户名和密码。Kettle默认的用户名/密码都是cluster。

文件中的密码可以利用Kettle自带的Encr工具来混淆。要生成一个Carte密码文件，使用-carte选项，像这个例子：

```
sh encr.sh -carte Password4Carte
OBF:1324324u432oj4324j312kj432j4321j4312j4312j43k24jc
```

使用文本编辑器，将返回的字符串追加到密码文件中用户名的后面：

```
Someuser : OBF : 1324324u432oj4324j312kj432j4321j4312j4312j43k24jc
```

OBF: 前缀告诉Carte这个字符串是被混淆了的，如果你不想混淆这个文件中的密码，可以像下面这样指定密码：

```
Someuser:Password4Carte
```

需要注意的是：密码是被混淆了，而不是被加密了。这个算法仅仅是让密码更难识别，但绝不是不能识别。如果一个软件能读取这个密码，必须假设别的软件也能读取这个密码。因此，应该给这个密码文件一些合适的权限。如果你能阻止别人未经授权访问这个文件，也会降低密码被破解的风险。

也可以使用JAAS (Java Authentication and Authorization Service) 来配置Carte的安全。此时需要定义如下两个系统变量（在kettle.properties中设置）。

- **loginmodule**name: 登录模块的名字。
- **java.security.auth.login.config**: 要使用的JAAS配置文件名。

JAAS用户域的名字是Kettle. 配置JAAS的具体细节超过了本书的范围。更多的信息可以在JAAS的主页找到: <http://java.sun.com/products/jaas/>。

16.2.6 服务

子服务器对外提供了一系列服务。表16-1列出了它定义的服务。这些服务位于Web服务的/kettlele/的URI下面。在我们的例子里, 就是<http://server:8181/kettle/>。所有的服务都有xml=Y选项, 这样可以返回XML, 客户端就可以解析。表16-1还说明了服务使用的类(包org.pentaho.di.www)。

表16-1 子服务器服务

| 服务名称 | 描述 | 参数 | Java类 |
|------------------|--|---|---|
| status | 返回所有转换和作业的状态 | | SlaveServerStatus |
| transStatus | 返回一个转换的状态并且列出所有步骤的状态 | name (转换名称); from line (增量日志的开始记录行) | SlaveServerTransStatus |
| prepareExecution | 准备转换, 完成所有步骤的初始化工作 | Name (转换名称) | WebResult |
| startExec | 执行转换 | Name (转换名称) | WebResult |
| startTrans | 一次性初始化和执行转换。虽然方便, 但是不适用在集群执行环境下, 因为初始化需要在集群上同时执行 | Name (转换名称) | WebResult |
| pauseTrans | 暂停或者恢复一个转换 | Name (转换名称) | WebResult |
| stopTrans | 终止一个转换的执行 | Name (转换名称) | WebResult |
| addTrans | 向子服务器中添加一个转换, 客户端需要提交XML形式的转换给Carte | | TransConfiguration WebResult |
| allocateSocket | 在子服务器上分配一个服务器套接字 更多内容请参考本章后面的“集群转换” | | |
| sniffStep | 获取一个正在运行的转换中, 经过某个步骤的所有数据行 | Trans (转换名称); Step (步骤名称); Copy (步骤的拷贝号); Lines (获取行数); Type (输入还是输出) | <step-sniff> XML包含了一个RowMeta对象以及一组序列化的数据行 |
| startJob | 开始执行作业 | Name (作业名称) | WebResult |
| stopJob | 终止执行作业 | Name (作业名称) | WebResult |
| addJob | 向子服务器中添加一个作业 客户端需要提交XML形式的作业给Carte | | JobConfiguration WebResult |

续表

| 服务名称 | 描述 | 参数 | Java类 |
|---------------|---|----------------------------------|---|
| jobStatus | 获取单个作业的状态并列出作业下所有作业项的状态 | Name（作业的名称）； From（增量日志的开始记录行） | SlaveServerJobStatus |
| registerSlave | 把一个子服务器注册到主服务器上（参考“集群转换”部分） 需要客户端把子服务器的XML提交给子服务器 | | SlaveServerDetection WebResult(reply) |
| getSlaves | 获得主服务器下的所有子服务器的列表 | | <SlaveServerDetections> 节点下包含了几个 SlaveServerDetection节点 |
| addExport | 把导出的.zip格式的作业或转换，传送给子服务器。文件保存为服务器的临时文件 客户端给Carte服务器提交zip文件的内容 这个方法总是返回XML | | WebResult里包含了临时文件的URL |

16.3 集群转换

集群技术可以用来水平扩展转换，使它们能以并行的方式运行在多台服务器上。转换的工作可以均分到不同的服务器上。本节将介绍怎样配置和执行一个转换，让其运行在多台机器上。

一个集群模式包括一个主服务器和多个子服务器，主服务器作为集群的控制器。简单地说，作为控制器的Carte服务器就是主服务器，其他的Carte服务器就是子服务器。

一个集群模式也包含元数据，元数据描述了主服务器和子服务器之间怎样传递数据。在Carte服务器之间通过TCP/IP套接字传递数据。之所以选择TCP/IP作为数据交换的方式，是因为Web Services比较慢，而且会带来不必要的性能开销。

注意：只有在集群模式中，才有主服务器和子服务器的概念。只要勾选子服务器的“是主服务器”复选框，一台子服务器就可以变成主服务器。不用给Carte传递任何特别的参数。

16.3.1 定义一个集群模式

在定义一个集群模式之前，需要先定义一些子服务器。（参考本章前面部分：定义子服务器。）定义完子服务器，右键单击Spoon里的“集群模式”节点，然后选择“新建”选项，如图16-16所示。

在配置窗口里设置集群模式的所有选项。至少选择一个主服务器作为控制器并选择一个或更多子服务器（如图16-17所示）。

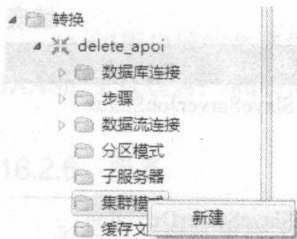


图16-16 创建一个新的集群模式

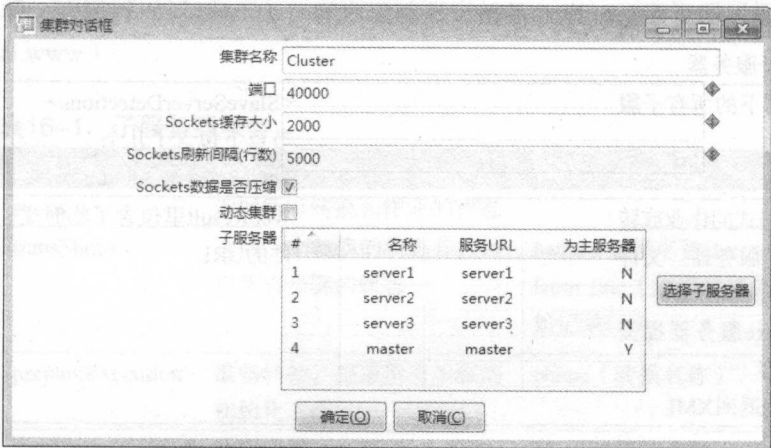


图16-17 “集群对话框”对话框

这里有几个重要的选项。

- **端口**：在服务器之间传递数据的最小的TCP/IP端口号。这是一个起始端口号。如果你的集群转换需要50个端口，从初始端口号到初始端口号+50之间的所有端口都会被使用。
- **Sockets缓存大小**：缓存大小用来使子服务器之间通信更平滑。不要把这个值设置得太大，否则数据传输过程可能会比较波动。
- **Sockets刷新闻隔（行数）**：因为进行Socket通信时，传送的数据行可能保存在Socket的缓存中，这里要设置一个flush间隔，缓存中的数据行积累到一定数量，转换引擎就会执行一个flush操作，强制把数据推送给对方服务器。这个参数的大小，取决于子服务器之间的网络速度和延迟。
- **Sockets数据是否压缩**：设置子服务器之间传输的数据是否需要压缩。对于相对较慢的网络（如10Mbps），可以设置这个选项。如果设置成“是”，将会导致集群转换变慢，因为压缩和解压数据流需要CPU时间。因此，在网络不是瓶颈时，最好不启用这个选项。
- **动态集群**：如果设置了这个选项，Kettle会在主服务器上自动搜寻子服务器列表，来构建集群。关于动态集群的更多内容请参考第17章。

16.3.2 设计集群转换

设计一个集群转换，需要先设计一个普通的转换。在转换里创建一个集群模式，然后选择你想要通过集群方式运行的步骤。右键单击这个步骤，选择集群。

例如，你可能想从一个共享网络里读取一个大文件，然后排序数据，再将数据写入另外一

个文件。如果你想在三个子服务器上并行执行读取和排序操作，设计的第一步如图16-18所示。

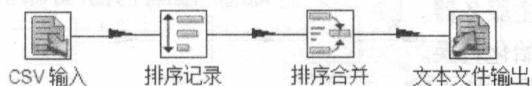


图16-18 一个普通的转换

下一步选择“CSV 输入”和“排序记录”步骤。在右键菜单中选择集群，选择完步骤要运行的集群模式后，转换将变成如图16-19所示的样子。

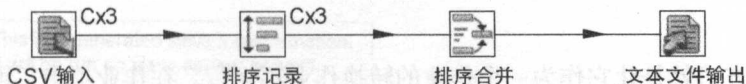


图16-19 一个集群转换

当你执行这个转换，所有被定义成集群运行（在图16-19中那些有“Cx3”标志）的步骤都在这个集群的子服务器上运行，而那些没有集群标识的步骤都在主服务器上运行。

注意：在图16-19中，“排序记录”步骤使用了三个不同的子服务器并行排序，所以就有三组排好序的数据行依次返回给主服务器。因为后面的步骤接收这三组数据，所以还要在后面的步骤里把这三组数据再排序，由“排序合并”步骤来完成这个工作，它从所有的输入步骤中一行一行地读取记录，然后进行多路合并排序。没有这个步骤，并行排序的结果是错误的。

如果转换中至少要有一个步骤被指定运行在一个集群上，这个转换才是一个集群转换。为了调试和开发，集群转换可以在Spoon的执行对话框中以非集群的方法执行。

注意：在一个转换中只能使用一个集群！

16.3.3 执行和监控

有两种方法可以运行一个集群转换。一个方法是在Spoon中选中“集群方式执行”选项，如图16-20所示。

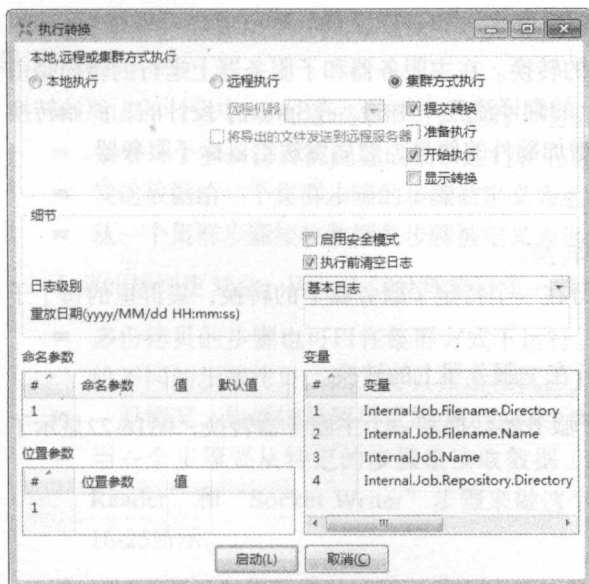


图16-20 选中“集群方式执行”

为了调试目的，可以使用下面几个集群选项。

- **提交转换：**提交生成的转换给子服务器和主服务器。
 - **准备执行：**在子服务器和主服务器上，初始化转换。
 - **开始执行：**在子服务器和主服务器上，执行转换。
 - **显示转换：**在Spoon里打开主服务器和子服务器上的转换，这样可以看到生成的转换。
- 关于更多子服务器和主服务器的转换，请看下一节。

请注意要完全运行一个转换，必须启用前三个选项。第四个选项非必需，仅方便你看到生成的转换。

另外一个运行集群转换的方法就是让它作为一个作业的转换作业项运行。在作业项中，可以启用“在集群模式下运行这个转换”选项，使得这个转换运行在一个集群上（如图16-21所示）。

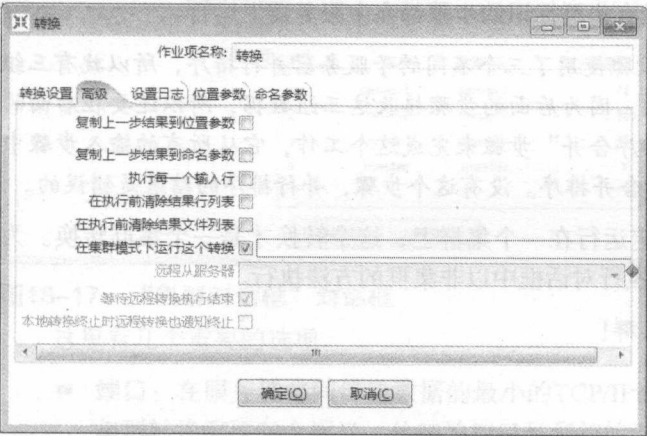


图16-21 通过一个作业项来运行集群转换

16.3.4 元数据转换

主服务器和子服务器上运行的并不是一样的转换。在主服务器和子服务器上运行的转换是由一个叫元数据转换（Metadata Transformations）的翻译流程产生的。在Spoon中设计的，原始转换的元数据，被切分成多个转换，重新组织，再增加额外的信息，然后发送给目标子服务器。

关于元数据转换，有以下三种类型的转换。

- **原始转换：**用户在Spoon中设计的集群转换。
- **子服务器转换：**它源自原始转换，运行在一个特定子服务器上的转换，集群里的每个子服务器都会有一个子服务器转换。
- **主服务器转换：**它源自原始转换，运行在主服务器上的转换。

在图16-19这个集群例子里，生成了三个子服务器转换和一个主服务器转换，图16-22显示了在我们例子里的主服务器转换。

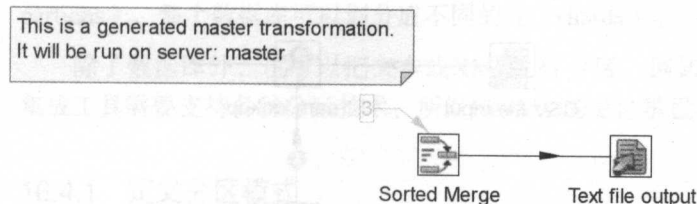


图16-22 一个主服务器转换

图16-23显示了子服务器转换。

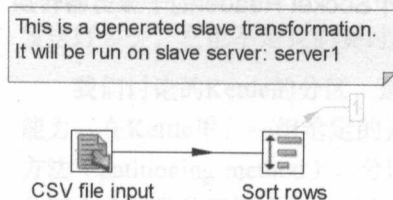


图16-23 一个子服务器转换

浅灰色编号的区域说明在步骤里有远程输入或输出连接，我们称之为远程步骤（Remote Steps）。在我们的例子中，有三个子服务器，每个子服务器把数据从“Sort rows”步骤发送到“Sorted Merge”步骤。这意味着三个“Sort rows”步骤都有一个远程输出步骤，并且“Sorted Merge”步骤有三个远程输入步骤。如果将鼠标悬置到这个浅灰色的矩形内，你将会获取更多关于这个远程步骤的信息，还有分配的端口号，如图16-24所示。

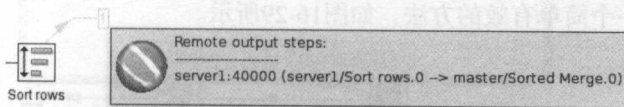


图16-24 远程步骤上的提示信息

规则

可以想象，操作这些元数据转换时，有很多可能性。让我们看看几个通用的规则，可以确保逻辑操作正确：

- 如果一个步骤被配置成集群方式运行，它会被复制到一个子服务器转换。
- 如果一个步骤没有被配置成集群运行，它会被复制到一个主服务器转换。
- 发送数据给一个集群步骤的步骤被定义为远程输出步骤（发送数据通过TCP/IP sockets）。
- 从一个集群步骤接收数据的步骤被定义为远程输入步骤（接收数据通过TCP/IP sockets）。

下面的规则更复杂，因为它们处理集群里一些更加复杂的功能。

- 多份拷贝的步骤也可以在集群方式下运行。在这种情况下，远程输入和输出步骤将分发给不同的步骤拷贝。因为拷贝在远程机器上运行，所以太多的步骤拷贝没有意义。
- 一般情况，集群转换要尽量简单，这样更容易分析生成的转换。
- 当一个步骤要从特定的步骤里读取数据（信息步骤），在生成的转换里使用“Socket Reader”和“Socket Writer”步骤来做这个工作，从特定的步骤里读取数据的转换如图16-25所示。

图16-26说明了在集群里的子服务器上执行的转换。图16-27说明了在集群里的主服务器上执行的转换。

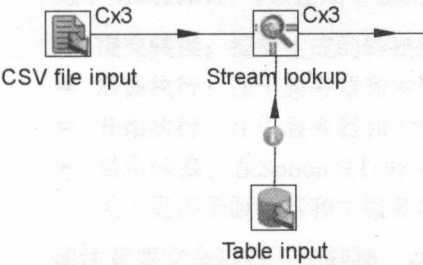


图16-25 信息步骤提供数据给集群步骤

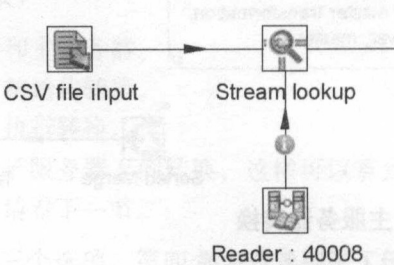


图16-26 带一个Socket Reader的子服务器转换

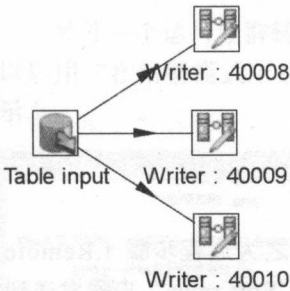


图16-27 主服务器分发数据给子服务器

仔细观察可以发现，“表输入”步骤使用“分发”方式把数据写到“Socket Writer”步骤里，传给子服务器，这不是我们想要的方式。在这种情形下，要使用“复制”的方式（见图16-28）。

当然在子服务器上读取三次数据也是一个简单有效的方法，如图16-29所示。

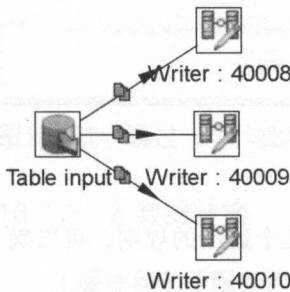


图16-28 使用复制的方式把数据复制到子服务器

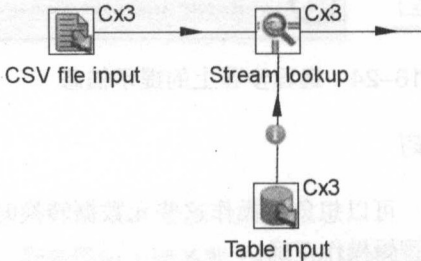


图16-29 在子服务器上获取数据

数据流水线

记得在本章前面提到数据流水线或数据的泳道：在Carte服务器之间交换的数据越多，转换就会越慢。理想情况下，你应该按照从头到尾并行执行的方式来组织你的数据。例如，处理100个XML文件会比处理一个单一的大文件更容易，因为在多份文件情况下数据能够被并行读取。

作为通用的规则，要使集群转换获取好的性能：尽量让转换简单，在同一子服务器上，尽可能在泳道里做更多的事情，以减少服务器之间的数据传输。

16.4 分区

分区是一个非常笼统的术语，广义地讲是拆分成多个部分。在数据集成和数据库方面，分区指拆分数据表或者把整个数据库分片（sharding）。表可以划分成不同的“表分区”（table

partions)，整个数据库可以划分成不同的片（shards）。

除了数据库外，也可以把文本或XML文件分区，例如按照每家商店或区域分区。由于数据集成工具需要支持各种分区技术，所以Kettle中的分区被设计成与源数据和目标数据无关。

16.4.1 定义分区模式

分区是Kettle转换引擎的核心，每当一个步骤把数据行使用“分发模式”发送给多个目标步骤时，实际就是在分区，分发模式的分区使用“轮询”的方式。实际上，这种方式并不比随机发送好多少，它也不是我们要讨论的一个分区方法。

我们讨论的Kettle的分区，是指Kettle可根据一个分区规则把数据发送到某个特定步骤拷贝的能力。在Kettle里，一组给定的分区集叫做分区模式（partitioning schema），规则本身叫做分区方法（partitioning method）。分区模式要么包含一组命名分区列表，要么简单地包含几个分区。分区方法不是分区模式的一部分。

图16-30是一个创建分区模式的例子，定义了一个包含两个分区（A和B）的分区模式。

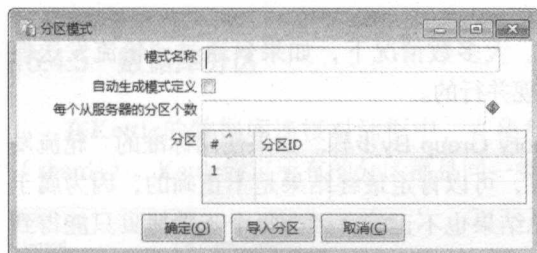


图16-30 “分区模式”对话框

定义完这个分区模式，在转换里，你就可以把它应用到一个步骤里，同时要设置分区方法。在步骤右键菜单中选择分区选项时，会弹出一个对话框让你选择使用哪个分区方法（如图16-31所示），分区方法可以是这些中的一个。

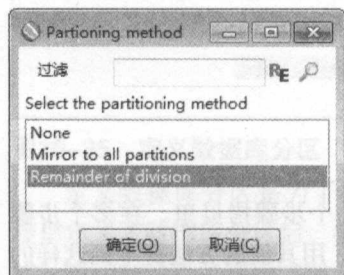


图16-31 选择分区方法

- **None**：不使用分区，只使用标准的“分发模式”或“复制模式”规则。
- **Mirror to all partitions**：在“数据库分区”部分，我们再描述这个方法。
- **Remainder of division**：这是Kettle标准的分区方法。Kettle用分区字段的整数类型的值（如果是其他数据类型，将使用这个类型的校验值）除以分区数目，产生的余数用来决定数据行发到哪个分区。例如在一个数据行里，分区字段的值是73，如果分区模式里有三个分区，这个数据行属于分区1。分区字段的值是30，这条记录属于分区0，分区字段的值是14，这条记录属于分区2。

- 通过插件实现分区方法：“分区模式”对话框里没有这个选项。具体参考第23章的一个分区插件例子。

在这个例子里，我们选择AB分区模式，如图16-32所示。

此时，会弹出一个对话框，可以在对话框里设置分区方法需要的参数。在我们的例子里，我们需要指定一个分区字段（如图16-33所示）。

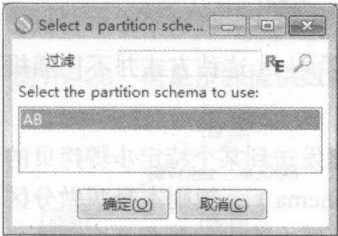


图16-32 选择分区模式



图16-33 指定基于它分区的字段

16.4.2 分区的目标

使用分区的目标是为了提高转换的并行程度。大多数情况下，如果只通过“轮流发送模式”把数据发送到多个步骤拷贝上，是根本不能实现并行的。

例如Group By步骤，为了简单，我们使用Memory Group By步骤。如果使用标准的“轮流发送模式”，把数据轮流发送到这个步骤的多份拷贝，可以肯定最终结果是不正确的，因为属于某一个组的记录可能在任意一个步骤拷贝里。汇总结果也不正确，因为每个步骤拷贝只能得到一个分组里的部分数据。

如果使用分区的方式来做并行分组，考虑下面的例子，有一个包含客户数据的文本文件，你想计算每个“州”的邮政编码个数（见图16-34）。

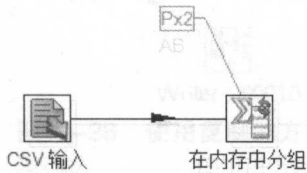


图16-34 一个分区例子

在州这个字段上做分区。可以确保每个州的数据发送到了同一个步骤拷贝里。在多个步骤拷贝的情况下，使用Memory Group By步骤也能计算出正确的结果，用其他的方法得不到这样的结果。

另外一个使用分区的原因是，如果你想并行执行“数据库查询”和“维度查询更新”步骤。如果你在这些步骤上应用分区，有可能提高缓存查询的命中率。因为有同样键的数据记录行出现在同样的步骤拷贝里，而且要查询的值在内存中的可能性也更大。

16.4.3 实现分区

在Kettle里实现分区很简单：对于每个分区步骤，Kettle会根据你选择的分区方法启动多个

步骤拷贝。如果定义五个分区，就会有五个步骤拷贝。分区步骤的前一个步骤（图16-34中的“CSV文件输入”步骤）做重分区的工作。当一个步骤里数据没有分区，这个步骤把数据发送到一个分区步骤的时候，就是在做重分区的工作。使用一种分区模式分区的步骤把数据发送给使用另一个分区模式的步骤，也会做重新分区的工作。

16.4.4 内部变量

为了方便处理分区数据，Kettle定义了一些内部变量。

- `${Internal.Step.Partition.ID}`：这个变量定义了步骤拷贝所属的分区ID或名称，在一个分区格式里面，可以用来读取或写入外部数据到带分区的Kettle步骤。
- `${Internal.Step.Partition.Number}`：这个变量定义了分区编号从0到分区数减一。

例如，数据已经被分区到N个文本文件中，（文件名从 file-0 到 file-N，N是分区数减一），可以创建一个“CSV文本文件”步骤，文件名定义为 file-`${Internal.Step.Partition.Number}`.csv，每一个步骤拷贝只读取属于它的分区数据文件。

16.4.5 数据库分区

在Kettle的数据库连接对话框中，在集群标签下，可定义数据库分区（partitions）或碎片（shards）。Kettle假定所有的分区都是同一类型的数据库（见图16-35）。

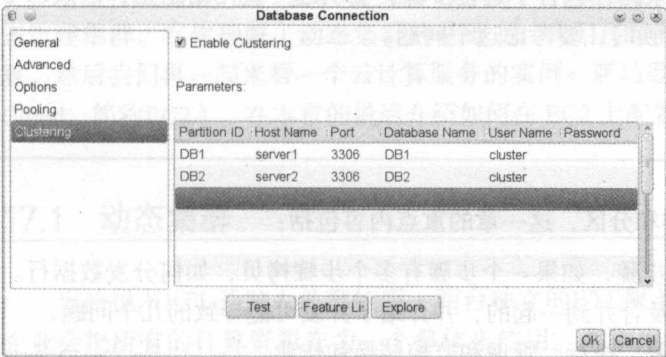


图16-35 定义数据库分区

定义数据库分区是为了把一个Kettle分区的数据写入到数据库分区，或相反。在数据库连接里定义了数据库分区后，实际就创建了一个分区模式。在创建Kettle分区时，可以在“分区模式”对话框中直接使用“导入分区”按钮（选择数据库分区模式）。

现在就可以在任何使用数据库的步骤中应用这个分区模式了。每个数据库分区都会有一个步骤拷贝，步骤拷贝直接连接到分区的物理数据库，数据库分区名字和步骤分区的名字相同。

图16-36是在两个不同数据库分区上并行执行一个查询，然后数据被并行发送到下面的两个步骤拷贝中，用来计算。

其他数据库相关的步骤也和上面的例子类似，这样多个数据库可以并行处理。另外“Mirror to all partitions”分区方法可以并行将同样的数据写入多种数据库分区。这样在做数据库查询时，就可以把查询表的数据同时复制到多个数据库分区中，而不用再建立多个数据库连接。

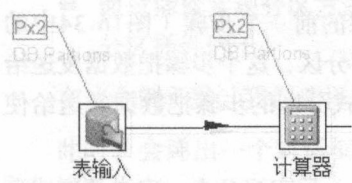


图16-36 读取数据库分区

16.4.6 集群转换中的分区

在一个转换里，如果定义了很多分区，转换里的步骤拷贝数会急剧增长。为了解决这个问题，就需要把分区分散到集群中的子服务器中。

在转换执行过程中，分区平均分配给各个子服务器。如果你使用静态分区列表的方式定义了一个分区模式，在运行时，那些分区将会被平均分配到子服务器上。这里有一个限制，分区的数量必须大于或等于子服务器的数量，通常是子服务器的整数倍（ $slaves \times 2$ ， $slaves \times 3$ ）。有一个解决这个问题的简单配置方法，就是指定每台子服务器上的分区数，这样在运行时可以动态创建分区模式，不用事先指定分区列表，如图16-30所示。

记住如果在集群转换里使用了分区步骤，数据需要跨子服务器重新分区，这会导致相当多的数据通信。例如，你有10台子服务器的一个集群，步骤A也有10份拷贝，但下面的步骤B设置为在每个子服务器上运行3个分区，这就需要创建 10×30 条数据路径，与图16-7中的例子相似。这些数据流向路径中的 $10 \times 30 - 30 = 270$ 条路径包含了远程步骤，会引起一些网络阻塞，以及CPU和内存的消耗，在设计带分区的集群转换时，要考虑这个问题。

16.5 小结

- 这一章介绍了转换的多线程、集群和分区，这一章的重点内容包括：
- 介绍了一个转换如何并行执行步骤，如果一个步骤有多个步骤拷贝，如何分发数据行。介绍了数据行是如何被分发以及合并到一起的，并介绍了并发可能导致的几个问题。
 - 介绍了如何在远程服务器上部署、执行、管理和监控转换和作业。
 - 深入介绍了如何使用多台子服务器构建一个集群，如何构建转换来利用这些子服务器资源。
 - 最后介绍了如何利用 Kettle分区来并行执行步骤，如何提高查询命中率。如何使用分区变量来并行处理文本文件，如何使用数据库分区模式来并行处理数据库的读写操作。

第17章 云计算中的动态集群

本章将继续第16章的Kettle 集群的介绍，本章主要讲述如何不使用固定数量的服务器来动态组建集群。在你理解了动态集群的基础上，我们再看看动态的资源集合，也就是常说的云计算。然后我们再一起来看一个云计算服务的实例：亚马逊弹性云计算（Amazon Elastic Compute Cloud，简称EC2）。在本章的最后介绍如何在 EC2 上配置一个Kettle 集群服务。

17.1 动态集群

当前很多ETL开发人员都使用一两台独立的ETL服务器来工作，但随着技术发展，更多的企业会把所有的计算资源作为一个整体来使用。本章为你介绍Kettle集群如何动态利用计算资源池。

在云计算和虚拟机这两个词流行之前，类似SETI@Home这样的一些初步应用已经在动态使用计算机的资源。SETI@Home是比较早的一种分布式动态集群；全世界的人们都可以把自己的计算机资源贡献出来，来帮助完成外星文明搜索（Search for Extra Terrestrial Intelligence）这一项目。SETI@Home是动态进行配置的，因为参加计算的节点总是处在变化中。事实上，SETI@Home是以屏保的方式实现的，所以事先不可能知道有多少机器在这个集群里。

随着云计算和虚拟机的发展，现在可以获得低价的计算资源。所以Kettle 也开始支持动态变更集群设置。

在Kettle的动态集群模式里，你只要定义一个主节点。子节点并不用如同上一章描述的那样在集群模式里定义。子节点会自动把自己注册到主节点。通过这种方式主节点可以在任何时刻知道集群的配置。主节点也会定时检查子节点是否还可用，如果子节点不再可用，主节点会把

子节点从集群中移除。

创建动态集群模式和创建普通集群模式类似（见第16章“定义集群模式”）。可以在“集群对话框”里选中“动态集群”来组建一个动态集群，如图17-1所示。

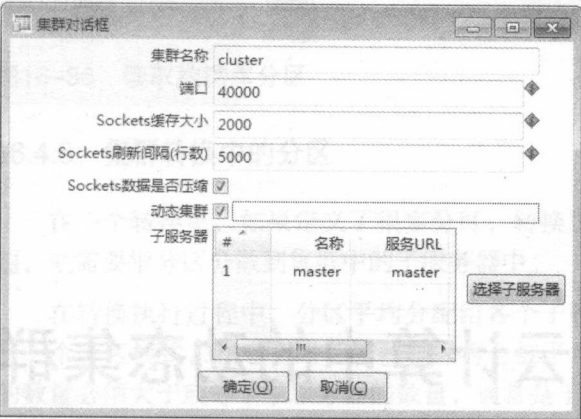


图17-1 创建一个动态集群模式

17.1.1 建立动态集群

要建立动态集群，首先在主节点上启动Carte程序。理想情况下，主节点应该是网络范围内固定的其他节点都能访问到的节点。一般情况下，主节点有一个固定IP或主机名。因为子节点需要和主节点通信，子节点必须要在启动前知道主节点的位置。

举个例子，在Kettle 4.x 发布包的pwd/ 目录下有一个主节点的最小配置文件。文件名是carte-config-master-8080.xml，内容如下：

```
<slave_config>

  <slaveserver>
    <name>master1</name>
    <hostname>localhost</hostname>
    <port>8080</port>
    <master>Y</master>
  </slaveserver>
</slave_config>
```

在这个配置文件中还可以设置更多的日志参数，这已经在前面章节介绍过。对Carte程序来说，主节点像其他服务器一样，它也是一个服务器。在UNIX 下，执行下面的命令：

```
sh carte.sh pwd/carte-config-master-8080.xml
```

在Windows 下，执行下面的命令：

```
Carte.bat pwd/carte-config-master-8080.xml
```

动态集群的主要特点在于子节点，而不是主节点。下面我们看看子节点如何配置，如何注册到主节点上。首先要配置这个子节点是否需要向主节点报告，它如何连接到主节点，连接到主节点的用户名和密码。子节点的配置文件如下：

```
<slave_config>
```

```

<masters>
  <slaveserver>
    <name>master1</name>
    <hostname>localhost</hostname>
    <port>8080</port>
    <username>cluster</username>
    <password>cluster</password>
    <master>Y</master>
  </slaveserver>
</masters>

<report_to_masters>Y</report_to_masters>

<slaveserver>
  <name>slave1-8081</name>
  <hostname>localhost</hostname>
  <port>8081</port>
  <username>cluster</username>
  <password>cluster</password>
  <master>N</master>
</slaveserver>
</slave_config>

```

在配置文件的<masters>部分，可以指定一个或多个主节点，子节点要向这些指定的主节点报告。尽管可以有多个主节点，但目前在Kettle 4.0里还不支持动态负载和故障切换。注意你需要同时指定主节点和子节点的用户名和密码。因为子节点要向主节点注册，而主节点要定时检查子节点是否可用。

最后，<report_to_master>选项设置子节点是否要向主节点报告。这个子节点的配置文件也在/pwd目录下，可以使用下面的命令启动：

```
sh carte.sh pwd/carte-config-8081.xml
```

启动后，不但可以在控制台看到通常普通的Web服务启动信息，还可以看到下面的信息：

```
Registered this slave server to master slave server[master1] on address
[localhost:8080]
```

此时，可以在Web浏览器中打开下面的页面：<http://carteserverhostname:8080/kettle/getSlaves/>。登录后，浏览器会以XML的格式返回请求：

```

<SlaveServerDetections>
  <SlaveServerDetection>
    <slaveserver>
      <name>Dynamic slave [localhost:8081]</name>
      <hostname>localhost</hostname>
      <port>8081</port>
      <webAppName/>
      <username>cluster</username>
      <password>Encrypted 2be98afc86aa7f2e4cblaa265cd86aac8</password>
      <proxy_hostname/>
      <proxy_port/>
      <non_proxy_hosts/>
    </slaveserver>
  </SlaveServerDetection>
</SlaveServerDetections>

```

```
<master>N</master>
</slaveserver>
<active>Y</active>
<last_active_date>2010/03/02 22:22:47.156</last_active_date>
<last_inactive_date/>
</SlaveServerDetection>
</SlaveServerDetections>
```

getSlaves 请求可以获取一个主节点下面的一组当前活动的或不活动的子节点列表。

17.1.2 使用动态集群

创建动态集群后，转换就可以使用这个集群。因为在设计时不知道集群里有多少个子节点，所以使用了动态集群的步骤用C×N这个标记来表示，其中N表示有N个子节点，如图17-2所示。

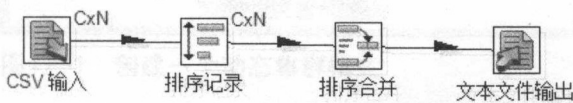


图17-2 以动态集群方式运行的步骤用 C × N来表示

只有在运行时才能获取子节点列表。如果在转换过程中有新的子节点加入，它们并不会加入正在运行的转换。如果在转换过程中子节点停止，子节点上的转换会停止，整个转换也会失败。

17.2 云计算

最近几年，我们看到在互联网上出现了一些云计算服务。它们都起源于 1999 年的Salesforce.com 网站的尝试。Salesforce.com是最早以网站形式提供企业级应用的公司之一。因为它的成功，很多其他公司也纷纷效仿，最后Web应用也给打上了软件即服务的标签（Software as a Service或SaaS）。

几年后的2002 年，亚马逊感到可以利用它的强大的计算资源来获得更多的收入，启动了亚马逊Web服务。最初，它只提供几个Web服务，如存储（S3）、亚马逊土耳其机器人（<http://www.mturk.com/>）服务。但是，只有在2006 年启动的弹性计算云（Elastic Computing Cloud，EC2）才真正开辟了一个新的市场，也就是所谓的基础设施即服务（Infrastructure as a Service，IaaS）。它可以使你以非常低的价格通过Web的方式来管理你自己的服务。

在EC2和类似服务之前，你还听过各种基于网格的服务，通常都是一些大公司提供的，像Oracle、Sun和IBM，它们也许诺以低价提供计算资源。这些网格计算平台和云计算的主要区别就是云计算使用虚拟资源来提供计算能力，而网格计算提供实际的机器。

云计算的优势非常明显：虚拟服务器可以很好地被管理，可以无限制地在它上面运行任何程序。可以在类似亚马逊 EC2的云计算服务器上完全独立地运行应用程序，可以做任何你想做的事情。可以创建一份虚拟机镜像，然后启动20份拷贝做同样的工作。因为可以远程管理，所以一般通过自动化的脚本来管理。这些优点再加上低价（一小时0.085美元起），使亚马逊的

EC2 迅速流行起来，用于运行各种服务。

因为可以根据镜像创建多份虚拟机实例，所以也可以通过这种方式创建动态集群。对于软件来说这也是一个挑战，因为软件要在事先不知道实例标识（如IP）的情况下，来使用这些实例。这也和动态集群的概念一致：在事先不知道包括哪些机器的情况下，来构建一个集群。

17.3 EC2

EC2是亚马逊为用户以虚拟机（VM）形式提供的真实的物理服务器阵列。由用户管理虚拟机。亚马逊的责任就是保证虚拟机在运行，EC2 用户的责任就是使用虚拟机。用户通过一些Web服务来管理虚拟机，这些 Web服务包括简单存储服务（Simple Storage Service, S3）、弹性块服务（Elastic Block Service, EBS）以及简单数据库和关系型数据库服务这些中间件。关于这些Web服务请参考<http://aws.amazon.com/>。

17.3.1 如何使用EC2

要使用EC2，你首先要在亚马逊 AWS 建立一个账号。在<http://aws.amazon.com> 注册，然后登录到EC2服务。

然后你要安装 EC2 命令行工具。下面的例子使用Linux 操作系统。注意这里的命令都是用Java 写的，在Linux、UNIX、Mac OS X以及Windows 操作系统下工作。关于如何安装命令行工具，参考：<http://docs.amazonwebservices.com/AmazonEC2/gsg/2006-02-26/setting-up-your-tools.html>。

17.3.2 成本

因为虚拟机实际也使用硬件，你也要支付使用虚拟机的费用（价格信息请见 <http://aws.amazon.com/ec2/#pricing>）。在编写本书的时候（2010年3月——译者注），一个最低价的服务实例的价格是0.085美元每小时，最高配置的“四倍超大内存”服务实例的价格是2.4美元每小时。

警告： 这些价格非常低，非常适合演示目的用，月使用费和年使用费也不能议价。因此在不需要的时候要关闭虚拟机。另外要注意最小计价单位是小时，即使只使用了5分钟，也要支付一小时的价格。

不只是计算资源要产生费用，在亚马逊S3上存储也有费用，价格从0.055美元到0.15美元每GB，取决于存储的使用（见 <http://aws.amazon.com/s3/#pricing>）。亚马逊EBS存储卷的价格是0.10美元每GB每个月。

最后，数据从EC2 实例传输到互联网也有费用，价格是0.15美元每GB和0.08美元每GB，取决于总的流量大小。数据传输到EC2的价格是0.10美元每GB。

17.3.3 自定义AMI

AMI是亚马逊机器镜像（Amazon Machine Image）的缩写。这个镜像包括了虚拟机运行需要的所有软件。也包括了32位和64位的多种操作系统。所以第一件事就是为虚拟机选择一个操

作系统。在我们这个例子里使用了Ubuntu服务器版, Ubuntu AMI 版本9.10 (Karmic Koala)。关于如何在EC2上使用, Ubuntu 有一个很好的教程, 见 <https://help.ubuntu.com/community/EC2StartersGuide>。

在本例中, 我们使用一个位于北弗吉尼亚的亚马逊的数据中心的最低价的服务实例 (32位)。AMI 编号是: ami-bb709dd2, 实际上其他带有新版本 Ubuntu的AMI 实例也可以运行。从下面的网址可以得到所有可用的AMI的一个列表: <http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171>。

如果想通过操作系统查找, 也可以找到其他操作系统。

在启动镜像之前, 你要确保在镜像启动后可以远程访问它。要能远程访问, 需要首先生成一个键对。键对可以让你远程通过安全shell的方式 (ssh) 来访问启动的实例。运行下面的命令可以生成一个键对:

```
ec2-add-keypair pentaho-keypair > pentaho-keypair.pem
Chmod 600 pentaho-keypair.pem
```

现在可以启动 Ubuntu服务实例了:

```
ec2-run-instances ami-bb709dd2 -k pentaho-keypair
```

执行下面的命令可以监控机器正在启动的状态:

```
ec2-describe-instance
```

几分钟后, 你就可以看到类似下面的一行:

```
INSTANCE      i-e7855a8c      ami-bb709dd2
ec2-72-44-56-194.compute-1.amazonaws.com      ip-10-245-30-146.ec2.internal
running pentaho-keypair 0      m1.small  2010-03-08T12:21:25+0000
us-east-1d      aki-5f15f636      ari-d5709dbc
```

正在运行的状态说明机器已经启动并在等待接受输入。现在你就可以使用ssh来连接了。使用ssh时要使用生成的键对授权:

```
ssh -i pentaho-keypair.pem ubuntu@ec2-72-44-56-194.compute-1.amazonaws.com
```

现在, 你已经登录到了远程机器, 可以做你想做的事情了。首先添加几个更新软件的资源:

```
sudo vi /etc/apt/sources.list
```

向文件里添加下面的资源:

```
deb http://us.archive.ubuntu.com/ubuntu/ karmic multiverse
deb-src http://us.archive.ubuntu.com/ubuntu/ karmic multiverse
deb http://us.archive.ubuntu.com/ubuntu/ karmic-updates multiverse
deb-src http://us.archive.ubuntu.com/ubuntu/ karmic-updates multiverse
```

执行下面的命令来更新软件:

```
sudo apt-get update
```

另外还需要安装下面几个软件。

- **Java:** 使用Java 6来运行 Carte。
- **Unzip:** 用来解压缩PDI 软件包。

■ **ec2-ami-tools**: 以后要用到的AMI 工具。

下面的命令用来安装上面的几个软件:

```
Sudo apt-get install sun-java6-jre unzip ec2-ami-tools
```

安装完以后, 可以下载最新的PDI 4.0或以上版本。如果想要社区版, 可以从sourceforge.net 的Pentaho 项目中下载。如果想要企业版, 可以通过Pentaho的技术支持门户。

获取社区版可以用下面的命令:

```
Wget http://sourceforge.net/projects/pentaho/files/Data  
Integration/4.0.0-stable/pdi-ce-4.0-stable.zip/download
```

也可以用 scp, 安全复制命令:

```
scp -i pentaho-keypair.pem pdi-ce-version.zip ubuntu@server  
... Amazonaws.com:/home/ubuntu
```

最后解压缩Kettle:

```
mkdir pdi  
cd pdi  
unzip ../pdi-ce-4.x.x.zip
```

通过上面的步骤就可以把 Kettle 安装在EC2 上了。

注意: 如果你使用了一些 Kettle 插件, 也要把插件放到 plugins 目录下面对应的子目录里。另外确保所有的服务器里都有这些插件。

接下来, 要确定这个服务器实例用于 Carte服务。因为对主节点和子节点要上传不同的Carte 配置文件, 可以在执行 ec2-run-instance 命令时, 通过-f 参数传递一个XML文件。这个文件最大不能超过16KB, 不过已经足够了。

在服务器实例上, 可以使用下面的Web Services调用来获取数据文件:

```
wget http://169.254.169.254/1.0/user-data -O /tmp/carte.xml
```

这个命令一般是放在脚本里, 作为开机的自动启动的脚本。就是说还需要创建一个启动文件, 文件名为 /etc/init.d/carte, 文件内容如下:

```
#!/bin/sh  
su ubuntu -c "/home/ubunbu/runCarte.sh"
```

这个启动脚本又执行了一个runCarte脚本, 注意不是用 root 用户执行, 而是用ubuntu用户执行。/home/ubunbu/runCarte.sh脚本的内容如下:

```
#!/bin/sh  
LOGFILE=/tmp/carte.log  
exec 2> $LOGFILE  
cd /home/ubuntu/pdi  
# Get the user data : a carte configuration file  
#  
wget http://169.254.169.254/1.0/user-data -O /tmp/carte.xml >> $LOGFILE  
# Now start carte...  
#  
Sh carte.sh /tmp/carte.xml >> $LOGFILE
```

该脚本把日志记录到 /tmp/carte.log文件。也可以根据你的需求设置其他日志文件。

最后再执行几个命令来设置可执行和开机启动:

```
sudo chmod +x /etc/init.d/carte
sudo chmod +x /home/ubuntu/runCarte.sh
sudo update-rc.d carte defaults
```

前两个命令使脚本可执行, 第三个命令设置开机启动。

17.3.4 打包新AMI

我们已经修改了原始的Ubuntu服务, 需要再创建一份镜像以备用。创建一份镜像的过程也称为打包, 使用ec2-bundle-vol 命令来做。关于打包的具体过程, 参考亚马逊EC2文档库 <http://docs.amazonwebservices.com/AmazonEC2/gsg/2006-06-26/>。下面简要描述打包过程。

首先将你的私钥和证书安全复制到EC2实例, 可以使用下面的命令:

```
scp -I pentaho-keypair.pem ~/.ec2/*.pem
ubuntu@ec2-72-44-56-194.compute-1.amazonaws.com:/tmp/
```

然后就可以打包镜像了:

```
sudo bash
ec2-bundle-vol -d /mnt/ -k /tmp/ -k /tmp/pk-your-pkfilename.pem
-u your-account -s 1536 -cert /tmp/cert-your-cert-file.pem
```

再上传这个AMI 镜像到S3, 以备将来使用:

```
ec2-upload-bundle -b kettle-book -m /mnt/image.manifest.xml
-a your-access-key -s your-secret-key
```

最后, 注册到你的本地机器 (不是EC2服务器):

```
ec2-register kettle-book/image.manifest.xml
IMAGE ami-bbfb14d2
```

最后获得的AMI 号, 可以用来启动 Carte服务。

17.3.5 中止AMI

如果不再需要你配置好的实例, 可以停止它。可以运行下面的命令停止实例 (注意把 your-instance-nr 替换成你自己服务实例的名字和编号):

```
ec2-terminate-instances i -your-instance-nr
```

可以使用 ec2-describe-instances 命令获得实例号。

17.3.6 运行主节点

要运行主节点, 需要创建一个 XML文件作为主节点的参数。这个文件就是前面讲过的Carte 的配置文件。我们把这个文件命名为 carte-master.xml:

```
<slave_config>
<max_log_lines>10000</max_log_lines>
<max_log_timeout_minutes >600</max_log_timeout_minutes >
<object_timeout_minutes>60</object_timeout_minutes>
```



```

<slaveserver>
  <name>master1</name>
  <network_interface>eth0</network_interface>
  <port>8080</port>
  <username>cluster</username>
  <password>cluster</password>
  <master>Y</master>
</slaveserver>

</slave_config>

```

你可能会发现，这里没有给Carte Web服务指定一个要监听的主机名。而是指定了一个网络接口，在EC2的Ubuntu服务器里，eth0 就是内部网络接口。

现在就可以启动一个AMI 实例了，把上面的配置文件作为参数：

```
ec2-run-instance ami-bbfb14d2 -f carte-master.xml -k pentaho-keypair
```

和以前一样，运行命令ec2-describe-instances，可以监控启动过程。几分钟后服务实例就启动了。

为了测试是否启动正确，打开Web浏览器，输入URL：

```
http://ec2-ip-address-etc.amazon.com:8080/
```

注意要给端口授权（类似于把开放的端口放入防火墙），以使其他人不能访问这个端口，授权命令：

```
ec2-authorize default -p 8080
```

17.3.7 运行子节点

主节点已经在运行了，现在可以启动多个子节点了，这些子节点可以向主节点报告。子节点的配置文件类似于：

```

<slave_config>

  <max_log_lines>10000</max_log_lines>
  <max_log_timeout_minutes>600</max_log_timeout_minutes>
  <object_timeout_minutes>60</object_timeout_minutes>

  <masters>
    <slaveserver>
      <name>master1</name>
      <hostname>internal-ip-address-of-the-master</hostname>
      <port>8080</port>
      <username>cluster</username>
      <password>cluster</password>
      <master>Y</master>
    </slaveserver>
  </masters>

  <slaveserver>

```

```
<name>Master</name>
<network_interface>eth0</network_interface>
<port>8080</port>
<username>cluster</username>
<password>cluster</password>
<master>Y</master>
</slaveserver>

</slave_config>
```

主节点的内部IP 地址是你唯一要替换的一个地方。可以通过执行ec2-describe-instances 命令来查看IP 地址。以 ip-开头以 .ec2.internal 结尾的服务名就是我们要的结果，中间部分就是IP 地址。例如我们得到一个服务名是ip-10-245-203-207.ec2.internal，那么IP 地址就是10.245.203.207。把这个地址放到上面的carte-slave.xml文件里。

现在我们就可以启动几个子节点了，启动命令：

```
ec2-run-instance ami-bbfb14d2 -f carte-slave.xml -k pentaho-keypair -n 3
```

选项“-n 3”说明启动了三个AMI实例。几分钟后就可以在浏览器中通过使用下面的getSlaves服务查看主节点下有几个子节点了：

```
http://ec2...amazon.com:8080/kettle/getSlaves
```

可以发现子节点一个接一个在列表里出现，而且子节点启动的速度并不完全一样。

17.3.8 使用EC2 集群

查看通过 /kettle/getSlaves服务获得的子节点，你会发现所有的子节点都是使用自己的内部地址注册到主节点上的。这种使用EC2 内部节点的方式很好，因为内部网络上的流量是免费的，而且很快。但是也随之产生一个问题，如果启动集群转换的客户端不能直接连接到子节点上，如何提交、运行、监控子节点运行的转换？为了解决这一问题和其他类似问题，Kettle 提供了一个“资源导出”的功能。这个功能可以导出一个作业或转换使用的所有资源到一个.zip文件中。因此，为了能在EC2 上运行集群转换，需要在一个作业里执行这个转换。然后使用“资源导出”将作业以及集群转换保存到一个.zip文件里，再把该文件传递给主节点——主节点再分发相应的资源到子节点，因为主节点和所有的子节点是连通的。当集群转换在主节点上运行时，所有子节点都是可见的，转换也能如期运行。

图17-3 展示了在“执行作业”对话框中，如何启用“资源导出”功能。

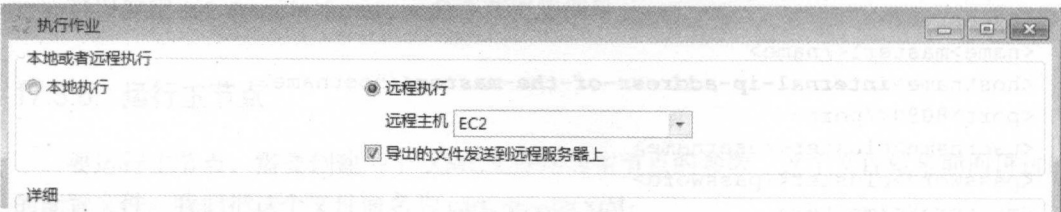


图17-3 作业导出给远程服务

也可以在“作业”作业项里启用同样的选项，如图17-4所示。

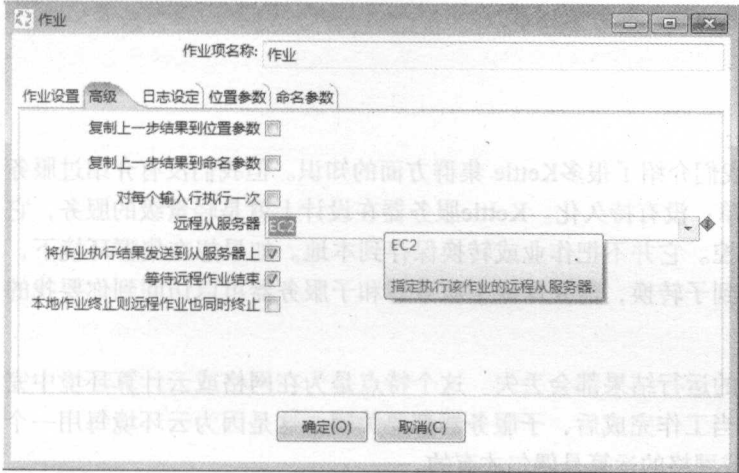


图17-4 在作业项里，将作业导出给远程服务

17.3.9 监控

尽管子节点是通过EC2 内部骨干网向主节点发送报告的，其实我们也可以通过子节点对外的IP 地址和指定端口（如 8080），来查看子节点的工作状态。

在动态集群里运行的转换，都被重命名了，在转换名中增加了“Dynamic Slave”字样，对应的服务URL 也重命名了。如果你要从日志中读取转换的信息，要注意这种转换名的变化。

当你在 Spoon 的集群服务器设置里找到了某个子服务器，将出现一个集群视图。在集群视图里，可以试试远程的行嗅探功能，如图17-5所示。通过这种方式可以很方便地看到各个服务器节点中转换里各个步骤的运行情况。

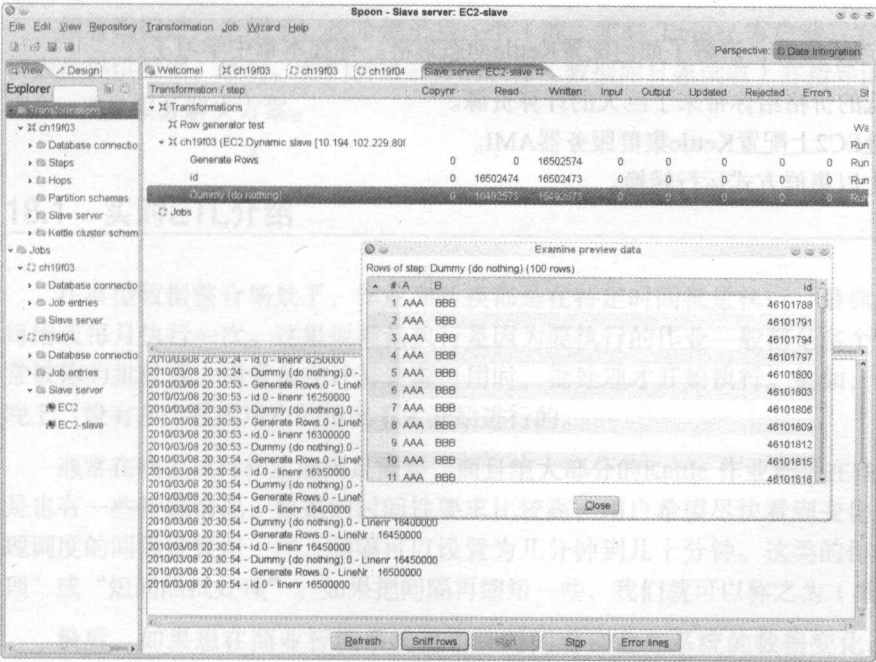


图17-5 远程行监控

关于行嗅探和实时监控的更多功能，参考第18章。

17.3.10 轻量原则和持久性

在前面各章节以及本章中，我们介绍了很多Kettle 集群方面的知识。但我们没有介绍过服务器数据的持久化问题。原因很简单：没有持久化。Kettle服务器在设计上就是轻量级的服务，它完全通过外部客户端来控制 and 监控。它并不把作业或转换保存到本地。如果想在集群环境下，从作业中找到转换或从转换中找到子转换，需要保证主服务器和子服务器可以访问到你要找的对象。

当一个服务器中止了，所有的运行结果都会丢失。这个特点是为在网格或云计算环境中完成数据整合工作而特别设计的。当工作完成后，子服务器都要关闭。这是因为云环境每用一个小时都要收费，而且需要基于云或网格的运算是偶尔才有的。

和Carte 一样，EC2 也不会保存任何数据。同样遵循着轻量级的设计原则，EC2只有在关闭和中止时才在本地保存信息。要保存的数据只是在AMI中捕获的数据。幸运的是，亚马逊还提供了弹性块服务（EBS），可以在EBS中创建文件系统，文件系统可以由主服务器和子服务器共享。EBS 上保存的信息和EC2 实例是分离的，将一直被保存，直到EBS 卷被删除。如果想给Kettle 集群的子服务器共享数据，EBS是最好的选择。

毋庸置疑，由于Carte服务和EC2实例都不保留数据会导致一些不便的情况。但是优点在于：如果你配置正确，可以关闭集群和所有的服务器，而不用考虑有何不良后果。就是说，一旦遇到麻烦，可以直接拔掉电源然后重启。

17.4 小结

本章更深入地阐述了集群，讲解了如何配置Kettle动态集群。你在本章中学习了：

- 云计算以很低的价格给你带来了巨大的计算资源。
- 如何在亚马逊EC2上配置Kettle集群服务器AMI。
- 如何在EC2 上以集群方式运行转换。

第18章 实时数据整合

本章我们来了解 Kettle 如何做实时的数据整合。首先来看部署实时数据整合时，我们主要面临的问题和需求。

然后，我们讲解 Kettle 转换引擎为什么适合实时 BI 解决方案。并讨论它的缺点和需要注意的问题。

接着我们讲一个例子，这个例子是一个（准）实时 Twitter 客户端，它不断更新数据库表为仪表盘提供数据。最后，我们讲第三方的软件（数据库日志读取）并指导读者如何在 Kettle 中实现 Java 消息服务解决方案。

18.1 实时ETL介绍

在典型数据整合场景下，作业和转换都是在特定时间批量执行。最典型的是在每天晚上、每周或每月执行一次。这里说批量执行是因为要执行的作业一般都是多个而且依次执行，也通常被称为批处理。通常是在计算资源可用时，批处理才开始执行。例如，数据仓库的更新是在晚上，没有多少在线用户的情况下才开始进行的。

通常在晚上执行作业能满足需要，而且绝大部分的 Kettle 作业都是在晚上进行的批处理。但是也有一些特殊情况，数据的时间性要求比较高，用户希望尽快看到变化的数据。可以把批处理调度的间隔设置短一些，间隔可以设置为几分钟到几十分钟。这类的作业可以称为“微批处理”或“短间隔批处理”。如果把间隔再缩短一些，我们就可以称之为（准）实时数据整合。

最后，如果想在商业智能系统里第一时间看到业务系统的数据变化，就要使用实时、连续、流式的数据整合。数据从源系统到目标系统就可以使用秒甚至毫秒来度量。实时数据整合

可用于生产系统的状态监控，即报警系统。当锅炉压力到了警戒值，监控人员需要尽快知道数据变化情况。在这种情况下就要使用实时数据整合，把从压力阀上获得的数据尽快传给监控人员。压力数据就可以通过简单仪表盘或进度图表现出来，称为实时商业智能解决方案。

18.1.1 实时处理面临的挑战

实时处理面临着很多挑战。最主要的挑战就是数据本身。最常见的挑战包括：数据比较、数据查询、求和/求平均、欺诈检测，以及各种各样的模式匹配。这些操作都是耗时的工作。在实时场景下，要考虑如何提高这些工作的效率。

除了数据本身的挑战外，当你把数据从源系统导入到目标系统时，你还要从数据源里抽取、传递和加载数据。这里任何一个环节都是对实时或准实时数据整合的挑战。

从数据源来看，当数据发生了变化，如何能最快知道数据如何变化非常重要，这样就可以把新增、删除或修改的数据抽取出来。第6章讲了对变化数据的抽取方法。下面列出几种常见的方法：

- 在数据中创建时间戳来判断变化的数据。
- 比较数据的快照。
- 使用触发器找到变化的数据。
- 通过关系型数据库的事务日志找到变化数据。

很明显，你想找的是一种低延时、无侵入性的方法。通常这意味着你要在数据库这一层做工作，也就是上面列表中的后两种方法。前两种方法相对被动一些，可能会给源系统增加负担或者延时较长。

正如你想象的，你从日志或者触发器获得的变化数据都是比较底层的数据。例如，当某个表里的某一行的某一列发生了变化，你可能只能得到一行里的变化的几列。如果想把变化的数据行导入到目标系统，就需要一些比较复杂的操作，因为你没有这一行全部的列，只有部分列。

最后总结一条就是：从数据库里获得的数据越靠底层，转换的复杂度就越高。有时候甚至要重新开发应用的业务逻辑来处理变化的数据。另外要记住，从关系型数据库日志中抽取变化的数据也没有任何标准可循。大多数关系型数据库产品都定义了自己的变化数据格式。获取的方法和输出通常也是不开放的，所以另一大挑战就是日志分析软件的成本很高。

昂贵的事务日志分析软件再加上对变化数据需要做的很多业务处理，使得从源系统获取数据的过程也变得耗时而高成本。

在输出的那一端，有一个系统也在处理无休止变化的数据。例如，OLAP系统会变得更复杂，因为要考虑增量数据，而不能用查询缓存。增量聚集也是一件比较复杂的事情。实时图表也容易出现，因为实时图表的数据是根据源系统的增量数据计算得到的。

18.1.2 需求

从前面的描述中可以看出实时数据整合系统是耗时而昂贵的。所以在做实时数据整合的实现之前，我们要看看有哪些需求。

所有实时数据整合类应用都有一个重要的共同的需求，就是数据的时效性，这些应用都希

望以最快的速度拿到变化的数据。还是继续上面那个压力罐的例子，如果用户是化工厂的操作员，计算机会给操作员实时报告某个压力罐的压力数据。这里要报告的是当时的压力数据而不是昨天晚上的压力数据。当压力超过警戒值时需要操作员采取紧急措施。

我们把这个例子分解一下，就会发现这个例子有三个主要部分：

- 捕获源系统里的数据变化，有一定的延时，以 S 来表示（压力传感器的刷新频率）。
- 变化数据展现给操作员，有一定的延时，以 P 来表示（仪表盘的刷新频率）。
- 采取相应措施（给压力罐减压），有一定的延迟，以 A 来表示。 A 就是从仪表盘读数据到采取措施之间的延迟时间。

从上面的简单需求分析可以看出，如果数据捕获（ S ）和展现（ P ）都是实时的，而采取行动的延迟时间比较长（ A ），这样的实时数据整合系统也没有太大意义。一个成功的实时系统应该能基于实时数据采取实时行动。在极端情况下，应该是一个不需要操作员的报警系统，在这种情况下实时数据整合是没有必要的，压力罐就要爆炸了。

因为实时（real-time）数据整合和商业智能的高成本，许多公司都在实践适时（right-time）商业智能解决方案这一概念。适时商业智能是按需要获取数据而不是在数据可访问后就实时地获取数据。适时商业智能策略使企业把精力放在数据传送时间上，而不是高成本的实时数据上。

18.2 基于流的转换

第2章曾经讲过，Kettle 转换的步骤之间使用流的方式传送数据。这种流的方式是通过步骤之间的缓存（跳）实现的，缓存有最大容量，最大容量也迫使数据行以流的方式在步骤间传送。不仅数据是以流的方式传送的，步骤也是也并行的方式来启动的，以利用多核计算机的性能优势。

在一个批转换里，输入步骤读取固定行数的数据。数据一次只读取一行，然后传递到下一个步骤里。下一个步骤再把数据传递给下一个步骤，直到所有的行都处理完毕。当所有的行都处理完毕，整个转换就结束了。在Kettle的转换架构中，缺少一个可以停止无休止运行的转换的机制。例如有一个系统不停地产生数据，但转换只开了很小的缓存，这种情况下要么读出数据，要么不读数据。正因为转换的这个缺陷，转换读取数据时可能导致转换一直在运行。

一个转换分为以下三个阶段。

- **初始阶段：**分配内存，打开文件，连接数据库，等等。
- **执行阶段：**执行数据行，执行所有的步骤。
- **清理阶段：**释放内存，关闭文件，关闭数据库连接，等等。

这种三阶段的处理方法适用于市面上的大部分ETL工具。但对于长时间连续运行的实时转换，还需要考虑超时、内存消耗和日志问题。

- **超时：**许多数据库都有连接超时选项。例如，一个已经连续运行了很长时间的转换，在一段时间内（如周末），转换要抽取的数据源没有任何更新的数据，这时就有数据库连接超时的风险。为避免这种风险，需要提前在数据库服务器和Kettle客户端设置连接超时参数。如何设置 Kettle数据库连接参数见第2章。

- **内存消耗**: 有几种情况可以导致在转换时消耗大量内存。一种情况是数据缓存, 缓存可以减少对数据库操作, 以提高“数据库查询”步骤、“维度查询/更新”步骤等的性能。当转换在不断接收新数据时, 一定要设置一个最大缓存大小, 或者确保内存的使用保持在一个平稳状态。另外, 注意在JavaScript等步骤里使用数据结构。注意不要使用持续消耗内存的数据结构, 如通常在“用户自定义Java 类”步骤里使用Map对象或者在Java脚本步骤里使用的关联数组。另一种情况是那些需要大内存的步骤, 如“排序”步骤, 需要接收到所有输入行后才进行排序, 在实时数据整合中, 因为要不断接收数据, 所以不能用“排序”步骤, 否则最终就会内存溢出。同样对于“流查询”步骤, 如果要查询的是一个永不停止的数据流, 最终也会内存溢出, 因为流查询步骤也需要在查询之前接收到流里所有的数据。这些步骤的共同特性就是要接收完所有数据才能开始工作, 所以只能在没有永不停止的数据流的情况下, 才能使用这些步骤。
- **日志**: 在以前版本的Kettle中, 一般是在转换或作业启动后, Kettle在数据库表中写一条日志记录。在转换结束后, Kettle 再去更新这条日志记录, 写入日志数据。如果转换是永不停止的, 这种做法实际就没有意义, 因为日志表中一直不会有实际的日志数据。为解决这一问题, 在Kettle 4.0中引入了间隔日志这一特性, 如图18-1所示。间隔日志会定时更新上述的日志记录, 这样可以定时通过查询日志表, 定时跟踪一个长时间运行的转换正在做什么工作。可以通过Spoon的历史视图或者Pentaho 企业版的控制台来查询日志表。在Kettle 4.0中, 还增加了日志写入行数的限制这一特性。这样有助于减少内存消耗并提升日志表里日志字段的可读性。除此之外, 从4.0 版本开始, 所有在一个Java虚拟机下运行的转换或作业都使用同一个中心日志缓存往日志表中写日志。Spoon、Carte、Pan、Kitchen等实例甚至 Pentaho BI Server, 都将采用这种方式写日志。在运行上面的这些程序时, 你都可以使用下面的环境变量(在KETTLE_HOME 目录下的kettle.properties文件中设置)。
 - KETTLE_MAX_LOG_SIZE_IN_LINES: 可以在内存中保存的最大日志行数。
 - KETTLE_MAX_LOG_TIMEOUT_IN_MINUTES: 一个日志行在被丢弃前可以在内存中保存的最长时间。通过设置这两个选项可以解决由日志导致的内存溢出问题。对于长时间运行的转换或作业, 日志会消耗大量的内存, 所以必须要设置这两个参数。

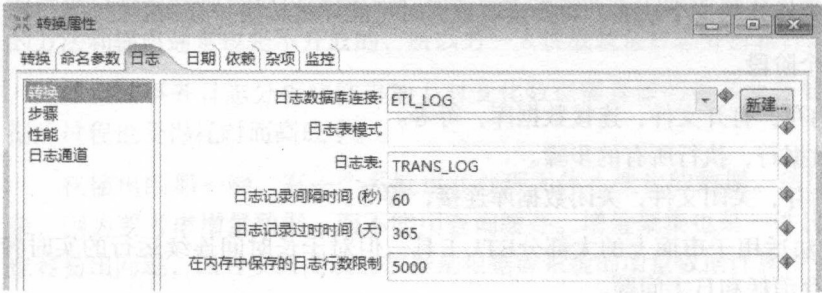


图18-1 间隔日志选项

18.2.1 一个基于流的转换实例

这个例子可以读取 Twitter的最近消息, 并在前端的仪表盘上显示。简单起见, 仪表盘会从本地数据库表中直接获取数据, 所以本地数据库表中的数据需要时刻变化。

为实现这个例子，要做的第一件就是如何从Twitter上采集信息。目前网上有几个Twitter的Java库。其中一个叫JTwitter（<http://www.winterwell.com/software/jtwitter.php>）。这个Java库非常小（160Kb），而且是LGPL协议，所以可以由Kettle使用。

这个例子在Kettle安装程序的samples/transformations目录下，文件名是User Defined Java Class – Real-time search on Twitter.ktr。

图18-2是一个由“采集Twitter信息”步骤和“更新数据库表”步骤组成的转换。

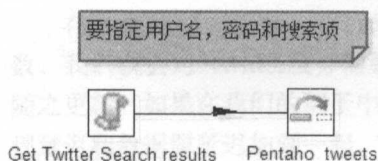


图18-2 不停地采集 Twitter 信息并保存在数据库表中

下面的例子展示了用户自定义Java类的步骤“获取Twitter搜索结果”中的代码。

```

import winterwell.jtwitter.*;

private Twitter twitter;
private long lastId;

public boolean processRow(StepMetaInterface smi, StepDataInterface sdi)
    throws KettleException
{
    java.util.List timeLine = twitter.search(getParameter("SEARCH"));
    logBasic("Received "+timeLine.size()+" tweets!");

    // First see if we need to look at any of the status reports
    // Determine the maximum ID and compare it to last times' status ID
    //
    long maxId=-1L;
    for (int i=0;i<timeLine.size();i++) {
        Twitter.Status status = (Twitter.Status)timeLine.get(i);
        if (maxId<status.getId()) {
            maxId=status.getId();
        }
    }

    // Do we have anything to do with this batch of statuses?
    //
    if (maxId>lastId) {
        // Process all the status reports...
        //
        for (int i=0;i<timeLine.size();i++) {
            Twitter.Status status = (Twitter.Status)timeLine.get(i);

            // If the id is recent, process it
            //
            if (status.getId()>lastId) {
                // New things to report...
            }
        }
    }
}
  
```

```

//
Object[] rowData = RowDataUtil.allocateRowData
    (data.outputRowMeta.size());
int index = 0;

rowData[index++] = status.createdAt;
rowData[index++] = Long.valueOf(status.getId());
rowData[index++] = status.inReplyToStatusId;
rowData[index++] = status.getText();
rowData[index++] = status.getUser().toString();
rowData[index++] = Boolean.valueOf(status.isFavorite());

putRow(data.outputRowMeta, rowData);
}
} else {
    // Wait for 30 seconds before retrying
    //
    int delay = Integer.parseInt(getParameter("DELAY"));
    for (int s=0;s<1000 && !isStopped();s++) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            // Ignore
        }
    }
}

lastId=maxId;

// Never end, keep running indefinitely, always return true
//
return true;
}

public boolean init(StepMetaInterface stepMetaInterface,
    StepDataInterface stepDataInterface) {
    if (super.init(stepMetaInterface, stepDataInterface)) {
        twitter = new Twitter(getParameter("USER"),
            getParameter("PASSWD"));
        lastId=-1;
        return true;
    }
    return false;
}
}

```

从上面代码中可以看到, 因为processRows()方法永远返回true, 而返回值true意味着“不断地运行该步骤, 不断执行该方法”。所以只有用户可以强制停止“Twitter采集”步骤。我们这里使用的Twitter客户端比较简单, 也不会做很多优化工作, 例如, 保持HTTP连接的打开状态。基本可以满足这个例子和大多数情况的需要。我们还可以在这个Twitter采集步骤里设置每次请求后的延迟参数, 如图18-3所示。

字段 参数 消息步骤 目标步骤

命名参数:

| # | 标签 | 值 | 描述 |
|---|--------|----------|-----------------------------------|
| 1 | USER | USERNAME | The twitter username |
| 2 | PASSWD | PASSWORD | The twitter password |
| 3 | SEARCH | pentaho | The search term |
| 4 | DELAY | 600 | Delay between searches in seconds |

图18-3 延迟参数

在本章的前面部分，我们定义了这个延迟实际就是数据源延迟。如果使用一个非常小的数，我们就会对 Twitter 服务器频繁地发出请求。如果发现了一个新的消息，数据库表也会很快随之更新。如果在我们的例子中，仪表盘 60 秒刷新一次，展现的最大延迟就是60 秒。我们把展现延迟和数据源延迟加到一起，就是用户采取行动的总延迟。

注意：我们不推荐对Web服务的连续查询，因为对免费 Twitter 服务器而言，这将是一种不必要的高负载。在测试上面例子的时候，请使用至少 60 秒的延迟。

18.2.2 调试

当实时转换在运行时，需要一种方法检查转换中的数据。可以使用预览，但预览只能看到前面的几行。如果转换已经运行了几天，你想看转换里的数据，你就需要其他的工具了。在 Kettle 4 版本里提供了两种方法来查看通过某一个步骤的数据行，我们也称之为行嗅探。第一种方法是在 Spoon 里，右键单击一个正在运行的步骤，在弹出菜单中选择“执行时嗅探”选项，然后选择要查看步骤的输入还是输出数据行。确定后，将出现一个预览行窗口，当数据从指定的步骤经过时会在该窗口中显示。

这种查看数据行的方式也可以用于远程执行（Carte）。在 Spoon 的“Slave Browser”标签下，可以单击“Sniff Test”按钮，来查看选中步骤的数据行，如图18-4所示。

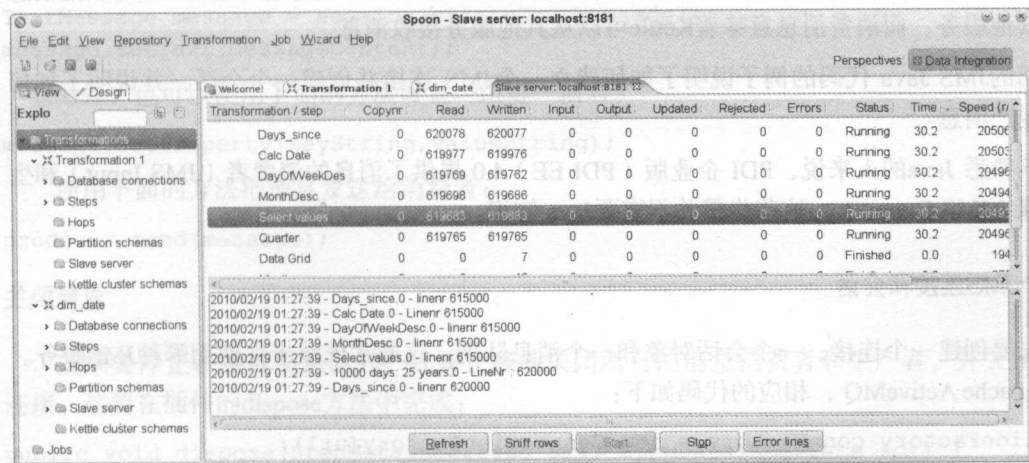


图18-4 在子服务器上嗅探数据行

18.2.3 第三方软件和实时整合

我们前面曾经介绍过，如果想处理某种实时数据流，可以利用第三方软件。例如，若想及

时获知Oracle数据库的变化,可以购买如LogMiner或Oracle Streams这类的软件。事实上,许多关系型数据库针对不同实时应用的场景都提供了事务日志读取软件。具体有哪些软件不在本书的范围内,可以把这些软件划分为以下两大类。

- **供外界查询变更数据的软件:** 这类软件可以通过标准或非标准的接口对外提供变更数据。例如SQLStream (<http://www.sqlstream.com>), 该公司的合伙人之一就是Julian Hyde, 他也是Pentaho Analysis (Mondrian) 的首席架构师。可以通过JDBC连接到SQLStream, 然后在“表输入”步骤中对实时数据流执行SQL语句。SQL查询将一直查询不会停止, 直到转换或者SQLStream服务器停止。相反, 也可以使用“表输出”步骤往SQLStream服务器中写入数据, 服务器将从一个不停止的数据流中读取这些数据。
- **只对第三方工具(如我们例子中的Kettle)传递变更数据的软件:** 在这种情况下, 每当有新数据时, 读取日志的工具将反复调用转换。下面的“Java 消息服务”就是这种软件的一个例子。

因为这种软件一般都针对一种特定数据库的, 所以对于你公司里的每一种数据库, 都要有一个独立的日志读取工具。

18.2.4 Java 消息服务

在数据整合和应用整合里一个比较流行的实时技术就是Java 消息服务, 即JMS。JMS是一个消息标准, 基于该标准Java应用可以传递消息。这些消息以分布的和异步的方式被传递。JMS 通常部署在J2EE的环境中, 作为一个服务。

JMS是一套完整的API, 它提供了在接收和发送消息时的各种功能, 并支持各种操作模式。发布和订阅模式可能是最常用的模式, 因为它允许多个消费者注册到同一个主题下。该模式还支持持久订阅, 持久订阅会在订阅者暂时不能连接的时候, 保留订阅者未读取的数据。

发布订阅模式包括两个主要场景: 生产消息和消费消息。生产消息意味着Kettle可以把消息发送给其他服务, 而消费消息意味着Kettle可以从其他服务接收消息。

下面的JMS Java 代码的例子说明了如何建立一个JMS 连接并创建一个会话, 并说明了如何消费和生产消息。

对不熟悉 Java的人来说, PDI 企业版(PDI EE) 4.0 提供了消息的消费者(JMS Input) 和生产者步骤(JMS Output), 这些步骤并不需要Java 知识。

创建一个JMS连接和会话

首先要创建一个连接、一个会话对象和一个消息队列。这通常依赖于你使用哪种JMS服务。例如对Apache ActiveMQ, 相应的代码如下:

```
ConnectionFactory connFactory = new ConnectionFactory(url);
Connection connection = connFactory.createConnection(username,password);
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
Destination destination = session.createQueue(queueName);
```

对其他种类的JMS服务而言, 通常只变化 ConnectionFactory 就可以。例如对OpenMQ, 要使用类似下面的代码, 不同处以粗体表示。


```
ConnectionFactory connFactory = new com.sun.messaging.  
    QueueConnectionFactory(url);  
Connection connection = connFactory.createConnection(username,password);  
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);  
Destination destination = session.createQueue(queueName);
```

把上面的代码加上异常处理,就可以直接用在“用户自定义Java类”步骤的init()方法里。

消费消息

要消费消息,首先需要先创建一个消费者对象:

```
MessageConsumer consumer = session.createConsumer(queueName);
```

同样,可以把这一行代码放到“用户自定义Java类”步骤或你自己开发的插件的init()方法里。创建完消费者对象后,要从 queueName 队列中读取消息,使用下面的代码:

```
Message message = consumer.receive();
```

把这行代码放在processRows()方法的开始处,就获得了一个消息对象。如果该方法返回了null,你就可以知道队列被关闭了,你也只能随之结束processRows方法,返回false。除了null这种情况外,processRows()可以一直返回true,如前面的Twitter的那个例子。拿到message对象后,可以把它转换成简单文本格式或XML格式(两种常用的格式)。可以把转换后的文本传递给下一个步骤或直接在当前代码中解析,如何做取决于你。

生产消息

对于生产消息,稍微修改一下前面的创建消费者的代码:

```
MessageProducer producer = session.createProducer(destination);
```

然后创建一个文本或XML消息, messageContent是一个消息字符串:

```
TextMessage message = session.createTextMessage();  
message.setText(messageContent);
```

另外,也可以使用下面的方法设置一组键—值对:

```
message.setProperty(keyString,valueString);
```

使用下面的方法把消息发送给消费者:

```
producer.send(message);
```

关闭消息

如果要停止转换或消息队列,你要正确地关闭所有的消息消费者和生产者,并关闭会话和连接。这要在插件的dispose方法中完成:

```
public void dispose(StepMetaInterface smi, StepDataInterface sdi) {  
    try {  
        messageProducer.close();  
        session.close();  
        connection.stop();  
        connection.close();  
    } catch (JMSEException e) {
```

```
        logError("Unable to close JMS objects", e);
    }

    super.dispose(smi, sdi);
}
```

在PDI中创建一个JMS步骤需要Java开发知识，而且根据实际需求不同，代码也有很大不同。不过任何熟悉JMS代码的人都可以通过插件的方式或用户自定义Java类的方式将自己的代码集成到Kettle中。

18.3 小结

- 本章主要介绍了实时数据整合并解释了实时ETL面临的挑战和需求。本章讨论了：
- 如何调整转换，使之适应不间断的数据流。重点要关注数据库超时、内存消耗和日志。
 - 一个连续读取Twitter消息的实际例子。
 - 如何调试一个长时间运行的转换。
 - 和第三方实时整合软件结合的可能性。
 - 如何使用Java 消息服务发送和接收消息。

19.1 Data Vault 模型介绍

Data Vault (DV) 模型是用于企业级数据仓库量身定制的建模方式。由 Dan (Damon) Chappell (http://www.damonchappell.com) 在 20 世纪 90 年代 Data Vault 模型获得了极大的成功。图 19-1 展示了

Dan Chappell 的 Data Vault 模型图。

Part

V

第五部分：高级主题

它是一种为企业级数据仓库量身定制的建模方式。

从字面上看，Data Vault 模型是一个数据仓库。它由表、视图、索引、存储过程、触发器、函数、包、角色、权限等组成。Data Vault 模型是一个数据仓库，它由表、视图、索引、存储过程、触发器、函数、包、角色、权限等组成。Data Vault 模型是一个数据仓库，它由表、视图、索引、存储过程、触发器、函数、包、角色、权限等组成。

本部分包括

第19章 Data Vault管理

第20章 处理复杂数据格式

第21章 Web Services

第22章 Kettle集成

第23章 扩展Kettle

CHAPTER

19

第19章 Data Vault管理

(Data Vault是一种建模方法, 这种建模方法介于3NF和星型模型之间, 这种建模方法保留所有历史数据, 无论数据在当时业务层面的对与错。中文有翻译为数据拱顶建模, 有翻译为数据保险箱建模, 但译者感觉都不很恰当, 所以没有翻译而保留原文。——译者注)

本章由DIKW 学院的Kasper de Graaf 编写, Kasper de Graaf是Data Vault 建模领域的专家。

数据仓库在20世纪90年代开始发展, Bill Inmon和Ralph Kimball分别阐述了他们对数据仓库的观点。这两种观点总体来说都是要创建一种环境, 以便支持分析和报表的应用。但是Bill Inmon和Ralph Kimball的观点有很大分歧, 他们之间的分歧带来了一场所谓的“大辩论”。

在我们看他们的分歧之前, 先看看这两种观点的相似之处。Inmon和Kimball都同意使用数据集市。数据集市就是面向终端用户的数据库。数据集市通常使用星型模型来建模(第4章有星型模型例子: 租赁店的星型模型), 并根据报表和分析的需求而优化。

他们的这两种观点的最大区别就是是否需要一个企业级的数据仓库(Enterprise Data Warehouse, EDW)。Inmon 认为需要一个EDW, Kimball 认为不需要。EDW本质上就是一个大的数据仓库, 包括了从企业各个数据库集成过来的所有的历史数据。EDW不能由终端用户直接访问。仅用来储存和报表相关的, 用于审计的各种历史数据。在Bill Inmon看来, EDW位于业务系统和数据集市之间, 也是数据集市的唯一数据来源。

注意: 关于数据仓库更详细的解释参考Pentaho Solutions一书的第6章。Wiley出版社, 2009年。

本章先介绍Data Vault 建模方法, 然后使用 sakila数据库介绍一个例子。

19.1 Data Vault 模型介绍

Data Vault (DV) 模型是用于企业级的数据仓库建模。由Dan Linstedt在20世纪90年代提出 (<http://www.danlinstedt.com>)。在最近几年, DataVault 模型获得了很多关注, 并在BI 社区里拥有了一批追随者。

Dan Linstedt将 Data Vault 模型定义如下:

Data Vault是面向细节, 可追踪历史的, 它是一组有连接关系的规范化的表的集合。这些表可以支持一个或多个业务功能, 它是一种综合了第三范式(3NF)和星型模型优点的建模方法。其设计理念是要满足企业对灵活性、可扩展性、一致性和对需求的适应性要求, 它是一种专为企业级数据仓库量身定制的建模方式。

<http://www.danlinstedt.com/about>

从上面的定义, 可以看出Data Vault 既是一种数据建模的方法论, 又是构建企业数据仓库的一种具体方法。Data Vault 模型由三个模块组成, 中心表、链接表、附属表。建模方法论里定义了Data Vault的组成部分和组成部分之间的交互方式。Data Vault的建模方法中还包括了最佳实践, 来指导构建企业数据仓库。例如, 业务规则应该在数据的下游实现, 就是说Data Vault 只按照业务数据的原样保存数据, 不做任何解释、过滤、清洗、转换。即使从不同数据源来的数据是自相矛盾的(例如同一个客户有不同的地址), Data Vault 模型不会遵照任何业务的规则, 如“以系统A的地址为准”。Data Vault 模型会保存两个不同版本的数据, 对数据的解释将推迟到整个架构的后一个阶段(数据集市)。

本章将讨论Data Vault模型的方法论, Data Vault的架构不在本书讨论范围内。

19.2 你是否需要Data Vault

尽管我们不能回答你是否需要一个Data Vault 解决方案, 但可以帮助你选择。我们认为选择依赖于下面的问题: 你是否需要一个企业数据仓库? (或你能否从一个企业数据仓库受益?) 在下一个段落中, 将阐述企业数据仓库的益处。为什么建立企业数据仓库是一个正确决定。

企业数据仓库可以清楚划分责任。企业数据仓库负责保存所有数据, 包括没有变更过的各种历史数据, 没有做过业务处理和质量改进的数据。这会给报表提供一个坚实的数据环境。报表唯一要做的事情就是把数据以符合适合决策支持的样式展现出来。也就是说展现出的数据会遵照当前的业务规则、清洗规则。因此, 在这一层的数据集市和星型模型将解释存储在数据仓库里的事实数据, 反映由当前规则解释的真实情况。

因为企业数据仓库保存原始的和完整的历史数据。所以以企业数据仓库为基础, 可以生成任意规则下的报表(如新的观点导致业务规则发生变化)。

最后, 还要提一下数据的可跟踪性。可跟踪性意味每个数据片段都可以被跟踪到源头。意味着每个报表都能解释其业务含义, 每个数据都经得起争论和推敲。

19.3 Data Vault的组成部分

下面的部分描述了Data Vault 模型的主要组成部分：中心表、链接表和附属表，以及它们之间的交互。还说明了Data Vault 模型的一般特点和处理过程。

19.3.1 中心表

中心表是包含业务主键的表，每个业务主键代表一个唯一实体。常见的中心表的例子有客户、雇员、产品、建筑物、资源和假期。

Data Vault的一个重要方面就是上面提到的业务主键。业务主键唯一标识组织内的某个业务实体。这意味着如果为某一个实体分配了一个业务主键，那么你组织内的每个人都应该知道这个主键代表着哪个实体。想一想你家里的钥匙，只能开你家的门，不能开别人家的门。业务主键的一个重要方面就是它有业务含义，用在日常业务的操作中，例如一个客户号、一个订单号、一个产品代码。用另一句话说，业务主键是业务角度的一个标识，而不是技术领域的标识（技术领域的标识通常是技术ID或主键）。

中心表（每一类实体都有自己的中心表）用来保存一个组织内的每个实体的业务主键。除一些元数据外，业务主键是中心表包含的所有内容。Data Vault 另一个吸引人的特点就是中心表和源是互相独立的。当一个业务主键被用在多个系统时，业务主键也只保留一份，其他的Data Vault的组件都是连接到这一个业务主键。这也就意味着企业数据都集成到了一起。

表19-1 列出了中心表中应该包含的列（所有字段都需要，不需要其他字段）。

表19-1 中心表的属性

| 属性 | 描述 |
|---------------|----------------------|
| 主键 | 系统生成的代理键，供内部使用 |
| 业务主键 | 唯一标识的业务单元，用于已知业务的源系统 |
| Load DTS | 数据第一次加载到EDW时系统生成的时间戳 |
| Record source | 定义了数据来源（例如源系统或表） |

表19-2 显示一个中心表hub_customer的例子。

表19-2 中心表hub_customer的例子

| id | BusinessKey | Load_DTS | Record_Source |
|----|--------------|---------------------|---------------|
| 1 | cst_24125670 | 12/17/2009-03:05:04 | sales.cust |
| 2 | cst_24125894 | 12/17/2009-03:05:04 | sales.cust |
| 3 | cst_67904567 | 12/17/2009-03:00:00 | mkt.customer |
| 4 | cst_67904568 | 12/17/2009-03:00:00 | mkt.customer |

注意表里的数据来源于两个系统，sales和mkt。从Record_Source字段可以看出，客户cst_24125670和cst_24125894 最初存在于sales系统，客户cst_67904567和cst_67904568最初存在于mkt系统。这些客户可能也在其他系统里，因为中心表的业务键是唯一的，所以从这里不能看出这些客户还在哪个系统里。

19.3.2 链接表

Data Vault 模型的第二个构成部分就是链接表。链接表是中心表之间的链接。一个链接表就意味着两个（或多个）中心表之间有关联。一个链接通常是一个外键，它代表着一种业务关联，如业务活动、业务主键之间的事务等。

表19-3 列出了链接表中的所有字段（所有字段都需要，不需要其他字段）。表19-4是一个链接表的例子。

表19-3 链接表的字段

| 属性 | 描述 |
|---------------|-----------------------|
| 主键 | 系统生成的代理键，供内部使用 |
| 外键 | 中心表的代理键（外键），关系在链接表中定义 |
| Load DTS | 数据第一次加载到EDW时系统生成的时间戳 |
| Record source | 定义了数据来源（例如源系统或表） |

表19-4 一个链接表的例子 lnk_customer_store

| id | hub_customer_id | hub_store_id | Load_DTS | Record_Source |
|----|-----------------|--------------|---------------------|---------------|
| 1 | 1 | 1 | 12/17/2009-03:05:04 | sales.cust |
| 2 | 2 | 1 | 12/17/2009-03:05:04 | sales.cust |
| 3 | 3 | 2 | 12/17/2009-03:00:00 | sales.cust |
| 4 | 4 | 3 | 12/17/2009-03:00:00 | sales.cust |

和第三范式的数据模型不同的是，Data Vault模型完全忽略了关系的关联类型（1:N;N:M;...）。在Data Vault 里，每个关系都是N:M方式关联（多对多）。这给模型带来了更大的灵活性；无论数据在不同的源系统中是什么关系，都可以保存在 Data Vault 模型中。

在链接表中，所有中心表外键的组合也可以形成一个唯一的键，这个唯一键我们通常称为链接表的业务主键，尽管在技术上看不一定百分之百准确。

19.3.3 附属表

到目前为止，我们已经讲了中心表和链接表。中心表代表了组织中的不同实体，如客户、商店、产品、雇员、订单，等等。链接表为中心表之间的数据建立起关系。有了中心表和链接表，我们就有了模型的骨架，剩下的事情就是要给这个骨架添加“血肉”，也就是附属表。

Data Vault 模型使用附属表来保存中心表和链接表的属性，包括所有历史变化数据。一个附属表总有一个（且唯一——一个）外键引用到中心表或链接表。

表19-5列出了附属表要包含的所有字段（所有字段都需要，不需要其他字段）。表19-6是一个附属表的例子。

表19-5 附属表的字段

| 属性 | 描述 |
|----|----------------|
| 主键 | 系统生成的代理键，供内部使用 |

续表

| 属性 | 描述 |
|---------------|----------------------|
| 外键 | 中心表或链接表的外键，由附属表中的行定义 |
| Load DTS | 数据第一次加载到EDW时系统生成的时间戳 |
| Load End DTS | 数据失效时的时间戳，时间戳已被改变 |
| Record source | 定义了数据来源（例如源系统或表） |
| 属性 {1... N} | 属性自身 |

注意：在Data Vault 模型的标准定义里，附属表的主键应该是附属表里参照到中心表或链接表的外键字段和Load DTS字段的组合。尽管这个定义是正确的，从技术角度考虑，我们最好还是增加一个只有一列的代理主键。使用只有一个列的代理主键执行 UPDATE 等SQL 语句比多列的主键要容易（可能也会更快）。另外，把外键列和Load_DTS 列联合建立唯一索引，也是一个好的习惯。

表19-6的数据包含客户的两个属性：City和Birthdate。如果源系统（sales.cust）中客户的一个属性发生了变化，附属表中就会增加一行，原来的一行的截止日期字段也会被更新。ID为4和56的数据行描述了cust_id是41的同一个客户在不同时间的属性。

表19-6 附属表sat_customer的例子

| ID | HUB_CUST_ID | LOAD_DTS | LOAD END DTS | RECORD_SOURCE | CITY | BIRTHDATE |
|-----|-------------|---------------------|---------------------|---------------|-----------|------------|
| 1 | 14 | 12/17/2009-03:05:04 | NULL | sales.cust | Paris | 05/09/2006 |
| 2 | 26 | 12/17/2009-03:05:04 | NULL | sales.cust | New York | 06/24/1998 |
| 3 | 37 | 12/18/2009-03:00:00 | NULL | sales.cust | London | 07/20/1963 |
| 4 | 41 | 12/18/2009-03:00:00 | 02/30/2010-03:00:00 | sales.cust | Amsterdam | 04/03/1941 |
| ... | ... | ... | ... | ... | ... | ... |
| 56 | 41 | 02/30/2010-03:00:00 | NULL | sales.cust | Utrecht | 04/03/1941 |

中心表和链接表都可以有多个附属表，事实上，我们也推荐多个附属表。考虑一下两个源系统sales和marketing中都包括了客户数据的情况。在两个源系统中同一个客户都有同一个主键（你比较幸运）。在这种情况下，最好为每个源系统都创建一个附属表。除了源系统之外还有其他一些原因需要拆分附属表，如属性的变化频率（变化快的属性和变化慢的属性）和数据类型。

注意，和链接表一样，附属表也不包含真正的业务主键，而是包含一个由外键列和Load DTS列构成的联合主键。

注意：一个纯粹的DV 模型的Load End DTS字段的值应该为 NULL，如表19-6所示，但对于查询来说，这并不是一个最好的方法。有时候，经常会使用如2999/12/31或者其他类似的未来日期。理想主义的开发人员经常会争辩这是一个不正确的日期，如果未来的日期不确定，那么这个日期就不应该有值。但从实践的角度，用户在业务上都应该有一个默认的日期。

19.3.4 Data Vault 特点

下面的列表并不是Data Vault 模型的完整的或正式的规范，只是用来说明Data Vault 模型的一些特性，用来加深读者对 DataVault 模型架构和建模技术方面的理解。

一个设计较好的Data Vault 模型应该具有以下特点：

- 所有数据都基于时间来存储，即使数据是低质量的数据，也不能在ETL过程中处理掉。
- 依赖越少越好。
- 和源系统越独立越好。
- 设计上适合变化。
 - 源系统中数据的变化。
 - 在不改变模型情况下的可扩展性。
- ETL作业完全可以重新启动。
- 数据完全可追踪。

19.3.5 构建 Data Vault 模型

构建 Data Vault 模型的几个基本步骤包括：

1. 中心表建模，关注业务主键。
2. 链接表建模，寻找链接表和中心表之间的事务关系（使用外键）。在给中心表和链接表建模过程中，更便于你了解公司的组织和业务模式。
3. 附属表建模，提供上下文类的信息，以完成Data Vault 建模。

注意：通常还会有第4步，“快照（Point-In-Time）表建模”。快照表（PIT）是一种具有帮助性质的冗余表，它基于附属表，可以使面向 Data Vault 模型的查询更简单。PIT表的介绍不在本书的范围之内，我们不在这里做深入介绍。

19.4 将Sakila的例子转换成Data Vault 模型

在第4章中，sakila数据库直接转换成了维度模型。在本章，先将sakila数据库转换为Data Vault 模型，然后再转换为和第4章一样的星型模型。

19.4.1 Sakila中心表

本例中，下面的几个实体：actor、category、customer、film、staff和store要转换成中心表是比较明显的。也有几个不太明显的实体，如inventory、payment和rental，它们可以作为链接表（因为它们的事务特性）。但如果把这些表转换为链接表，就会在链接表之间生成新的链接表，就是所谓的链接表的链接表。我们不推荐使用这种结构，所以上面这些实体将作为中心表。

最后要考虑一些地理信息表，如address、city、country和language 表是否要作为中心表。sakila的开发人员把这些表都作为独立的实体。但对你来说，这些表都是参考数据。对你的业务来说（DVD 租赁），这些表不能算是中心表实体（在后面会看到，这些表实际是附属表）。

表19-7 列出了所有的中心表。

注意：中心表、链接表、附属表的选择不是绝对的。如果想把address、city和country 作为实体表，你就可以把它们转换为实体表。你将发现 Data Vault 模型很灵活。

表19-7 Sakila的中心表

| 实体 | 业务主键 |
|---------------|---------------|
| hub_actor | actor_id |
| hub_category | category_name |
| hub_customer | customer_id |
| hub_film | film_id |
| hub_inventory | inventory_id |
| hub_payment | payment_id |
| hub_rental | rental_id |
| hub_staff | staff_id |
| hub_store | store_id |

19.4.2 Sakila 链接表

选完了中心表，就要选择链接表。这意味着你要分析中心表的事务和它们之间的关系。在源系统里的外键非常有助于这一步。

事务类型的表比较容易找到：payment和rental。事务类型的表是Sakila的业务核心。所以，即使payment和rental 已经在前一个步骤中被选为中心表，考虑到它们的事务特性，我们也要在这一步中把它们选为链接表。

其他的链接表就是关系类型的表，一个顾客通常只访问一个商店（离他最近的商店），在sakila源系统中，customer 表有一个外键 store_id来表示这种关系。就是一个典型的链接的例子，来说明Data Vault 模型和普通的数据模型的区别。源数据模型（sakila）定义了一种一对多的关系（一个客户有且仅一个商店）。Data Vault 模型定义了每个关系都是多对多的关系（一个顾客可以有多个商店）。所以需要有一个表示关系的表link_customer_store。

全部链接表见表19-8。

表19-8 Sakila 链接表

| 链接表 | 被链接的中心表 |
|--------------------------|--|
| link_film_actor | hub_film, hub_actor |
| link_film_category | hub_film, hub_category |
| link_customer_store | hub_customer, hub_store |
| link_inventory | hub_inventory, hub_film, hub_store |
| link_payment | hub_payment, hub_customer, hub_staff |
| link_payment_rental | hub_payment, hub_rental |
| link_rental | hub_customer, hub_inventory, hub_staff |
| link_staff_worksin_store | hub_staff, hub_store |
| link_store_manager | hub_staff, hub_store |

注意：在sakila数据库里，store 表里包含一个外键参照到 staff 表（manager_staff_id），说明了商店的经理。在staff表里也有一个外键参照到 store 表（store_id），说明这个员工属于哪个商店。这两个外键每个都转换成一个链接hub_staff和hub_store 表的链接表；这两个链接表都是多对多关系的表，所以看上去它们是一样的。

19.4.3 Sakila 附属表

中心表和链接表构成数据的框架，如图19-1所示。

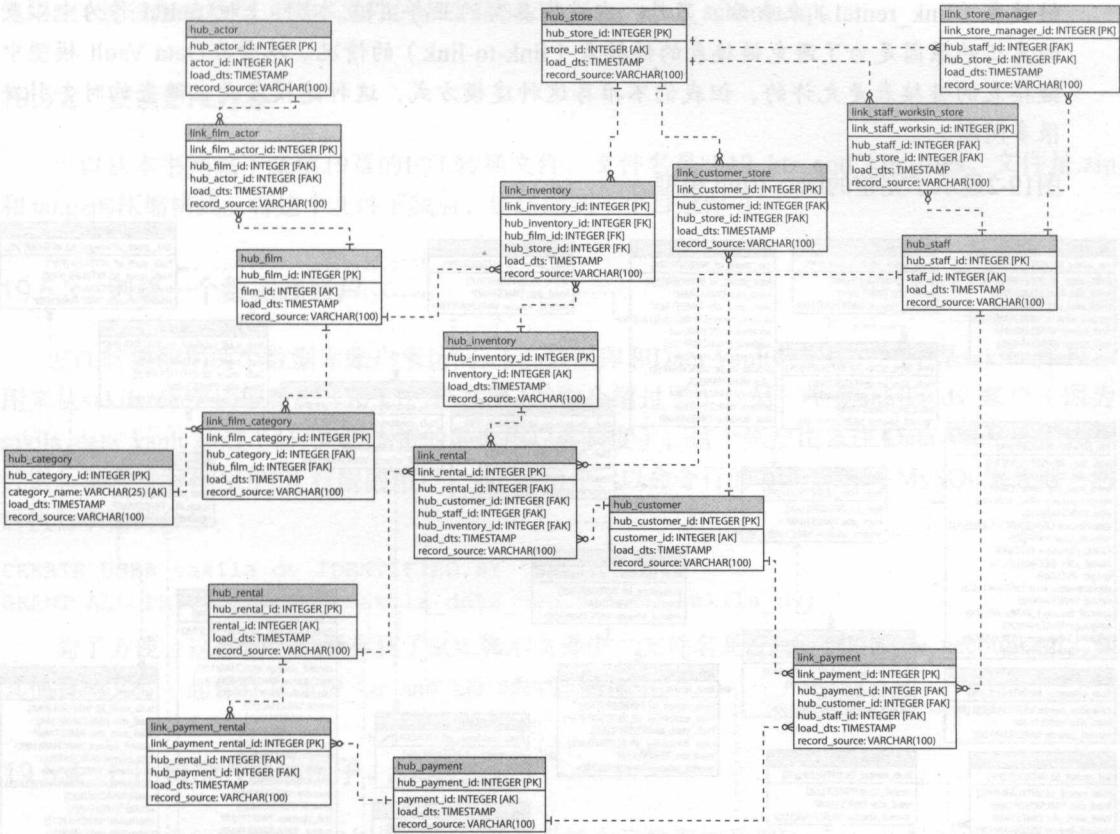


图19-1 Sakila Data Vault模型的框架

附属表将Data Vault数据模型补充完整，并为中心表和链接表补充属性。用在 Data Vault 模型里的所有源系统中的表的属性，最终都要放到Data Vault 模型中。

附属表的定义相当简单，所以我们直接来看表19-9 列出的所有附属表。

表19-9 Sakila附属表

| 附属表 | 描述 |
|--------------|--------------|
| sat_actor | hub_actor |
| sat_film | hub_film |
| sat_customer | hub_customer |
| sat_payment | hub_payment |
| sat_rental | hub_rental |

件，并验证这些脚本和转换文件可以正确打开。

19.5.1 安装Sakila Data Vault

先从本书网站下载Skaila Data Vault的SQL脚本。和Sakila例子数据库一样，脚本文件是以.zip和.tar.gz 压缩文件的方式提供的。

所以，Data Vault模型的安装过程和Sakila例子数据库的安装方式是相同的：解压缩文件，然后使用MySQL的SOURCE命令执行脚本。和Sakila例子数据库不同，脚本里不提供数据，我们会使用ETL过程加载数据。脚本文件的名称是Sakila_data_vault_schema.sql。

19.5.2 安装ETL方案

可以从本书网站下载第19章的ETL转换文件，文件名是ch19_ktr_and_kjb_files，文件是.zip和.tar.gz的压缩格式。将这个文件下载后，解压缩到一个目录即可。

19.5.3 创建一个数据库账户

ETL转换使用两个数据库账户来访问sakila数据库和Data Vault数据库：一个是sakila 账户，用来从sakila 库中读取数据（我们已经在第4章中介绍过了），另一个是sakila_dv 账户（因为sakila_data_vault 超过了MySQL 支持的最大用户名长度），这个账户用来往 Data Vault 库中读写数据。我们使用有 SUPER 权限的用户，如root用户，以命令行的方式登录到 MySQL服务器。然后执行下面的命令：

```
CREATE USER sakila_dv IDENTIFIED BY 'sakila_dv';
GRANT ALL PRIVILEGES ON sakila_data_vault.* TO sakila_dv;
```

为了方便，这些命令都保存到了SQL脚本文件中，文件名是create_sakila_dv_account.sql，和其他转换文件一起保存在ch19_ktr_and_kjb_files 压缩包中。

19.5.4 ETL解决方案的例子

做好数据库的准备后，我们再看如何从源系统中将数据加载到Data Vault 模型中。请记住，这只是一个简单的例子，只有一个源系统。

注意：在现实场景中，一般都需要一个数据缓冲区。但为了简明说明问题，在下面的例子里我们没有使用数据缓冲区。

Data Vault 模型的优点之一就是ETL设计和加载数据的过程可以复用。因为模型严格遵守规则，每个中心表的结构基本类似，链接表和附属表也都基本类似。这就是说加载过程非常类似，这样加载过程甚至是建模过程都有可能做到自动化。

因为上述的相似性，我们只讲述三种转换文件，分别是：简单中心表转换、简单链接表转换和简单附属表转换。

简单的中心表: hub_actor

加载 hub_actor 表的转换是hub_actor.ktr。转换如图19-3所示。

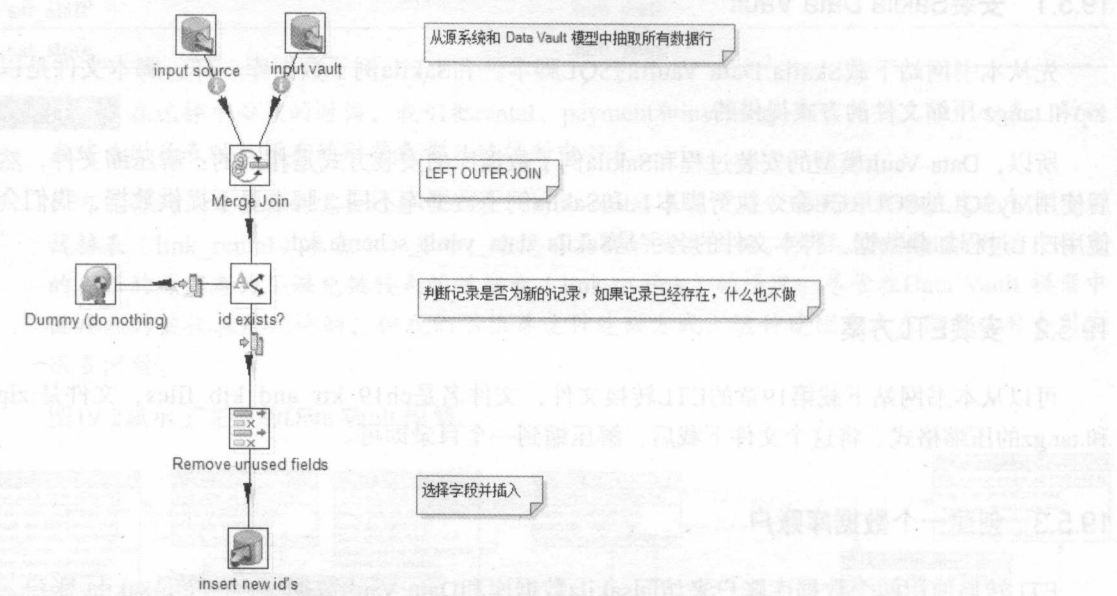


图19-3 hub_actor的转换

转换 hub_actor 负责插入Data Vault 模型中没有而源系统的 actor 表中有的所有演员实体（业务主键）。

下面简单介绍一下 19-3 转换中的主要步骤。

- input source: 该步骤在源库中执行下面的SQL 语句。

```
SELECT actor_id AS id
, 'sakila-db.actor' AS record_source
FROM actor
ORDER BY 1
```

这条SQL 语句从sakila 库中选择所有的actor_id（业务主键）。为了方便，通过SQL 语句直接把数据源定义为常量 sakila-db.actor。最后按照第一个字段 actor_id 进行排序。

- input vault: input vault步骤使用下面的SQL 语句从目标库中选择 actor_id。

```
SELECT actor_id AS id_existing
FROM hub_actor
ORDER BY 1
```

同样，类似上一个步骤，这个SQL查询也返回一组 actor_id。

- Merge Join: Merge Join步骤把前两个步骤的结果使用 LEFT OUTER JOIN的方式连接起来，类似下面的结果。

| id | record source | id_existing |
|----|-----------------|-------------|
| 1 | sakila-db.actor | 1 |
| 2 | sakila-db.actor | 2 |
| 3 | sakila-db.actor | 3 |
| 4 | sakila-db.actor | 4 |
| 5 | sakila-db.actor | NULL |
| 6 | sakila-db.actor | NULL |

这个列表说明源系统sakila包含六个actor_id，而Data Vault表hub_actor只包含了actor_id 1~4。很明显，actor_id 5和6是新插入到源系统中的数据，因此需要加载到Data Vault的模型中。

注意：你可能注意到从“input source”和“input vault”步骤到“Merge Join”步骤有两个带字母 i 的蓝色圆形图标。这些图标说明 Merge Join步骤期望接到的数据是排好序的。排序可以使用Sort Rows步骤。在我们的例子中，我们让数据库完成这些工作（使用 ORDER BY）。

- id exists?: 下一个步骤是Switch/Case步骤。id_existing字段值是NULL的数据行都被送到了Remove unused fields步骤，其他行则被送到了Dummy(do nothing)步骤，这些行将被忽略。
- Remove unused fields: 这个步骤移去无用的字段，只保留两个字段，即actor_id（以前的id字段）和record_source字段。
- Dummy(do nothing): 和步骤名一样，什么也不做。
- insert new ids: 最后的表输出步骤往 Data Vault 表hub_actor中插入数据行。

注意：除了actor_id和record_source字段，hub_actor 表还包含了两个其他字段：hub_actor_id和load_dts。这些字段的值是自动生成的。hub_actor_id是自增列，load_dts 有默认值CURRENT_TIMESTAMP，这样可以确保每插入一行数据都有一个时间戳。

如果你的数据库现在可以运行，也建立了Sakila_data_valut 库和相应的用户账号，你就可以直接运行转换，并将数据加载到hub_actor 表中。在普通的计算机上，这个转换几秒钟就可以执行完毕。

链接表例子: link_customer_store

将数据加载到 link_customer_store 表里的转换文件是link_customer_srote.ktr。转换如图19-4所示。

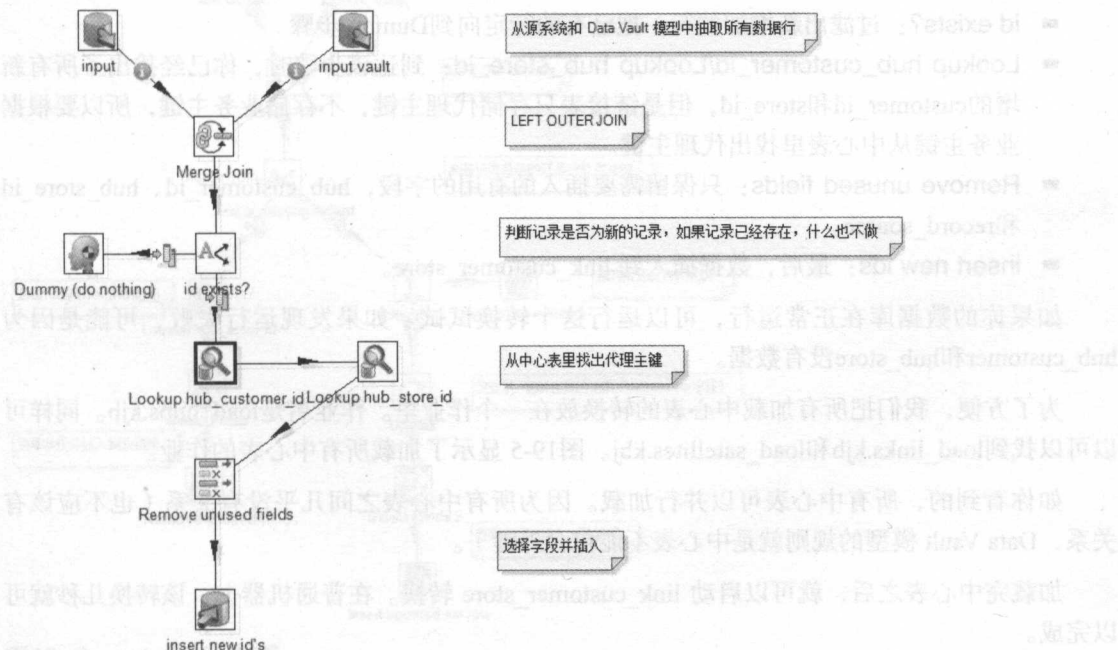


图19-4 link_customer_store 转换

link_customer_store 转换将以增量的方式往链接表里插入顾客和商店之间的所有关系。顾客和商店之间的关系就是顾客在哪个商店注册。

图19-4的主要步骤如下。

- **input:** input步骤对源数据库执行下面的SQL 语句。

```
SELECT    customer_id
,         store_id
,         'sakila-db.customer' AS record_source
FROM      customer
ORDER BY 1, 2
```

这个SQL语句从 sakila.customer表中选择每个customer_id和store_id的组合（业务主键）。

- **input vault:** 类似于中心表转换的input vault步骤，这个SQL 语句从Data Vault中选择所有customer_id和store_id的组合。因为这里的customer_id和store_id是业务主键，所有我们要把链接表link_customer_store和中心表hub_customer、hub_store 使用 inner join 链接起来：

```
SELECT      customer_id AS customer_id_vault
,           store_id AS store_id_vault
FROM        hub_customer
INNER JOIN  link_customer_store
USING      (hub_customer_id)
INNER JOIN  hub_store
USING      (hub_store_id)
ORDER BY    1, 2
```

customer_id和store_id 列在查询里做了重命名（加了_vault 后缀），用来区分和input步骤里过来的同名的两列。

- **Merge Join:** 同样类似于中心表的例子，Merge Join步骤对前面过来的两个来源的业务主键做 LEFT OUTER JOIN 操作。
- **id exists?:** 过滤出新增加的行，把已有的行定向到Dummy步骤。
- **Lookup hub_customer_id/Lookup hub_store_id:** 到达该步骤时，你已经找出了所有新增的customer_id和store_id，但是链接表只存储代理主键，不存储业务主键，所以要根据业务主键从中心表里找出代理主键。
- **Remove unused fields:** 只保留需要插入的有用的字段，hub_customer_id、hub_store_id 和record_source。
- **insert new ids:** 最后，数据插入到 link_customer_store。

如果你的数据库在正常运行，可以运行这个转换试试。如果发现运行失败，可能是因为hub_customer和hub_store没有数据。

为了方便，我们把所有加载中心表的转换放在一个作业里。作业名是load_hubs.kjb。同样可以找到load_links.kjb和load_satellites.kjb。图19-5 显示了加载所有中心表的作业。

如你看到的，所有中心表可以并行加载。因为所有中心表之间几乎没有关系（也不应该有关系，Data Vault 模型的规则就是中心表不能包含外键）。

加载完中心表之后，就可以启动 link_customer_store 转换。在普通机器上，该转换几秒就可以完成。

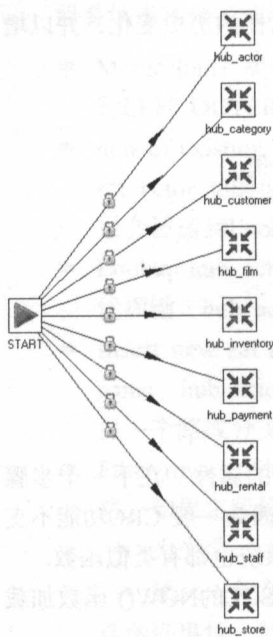
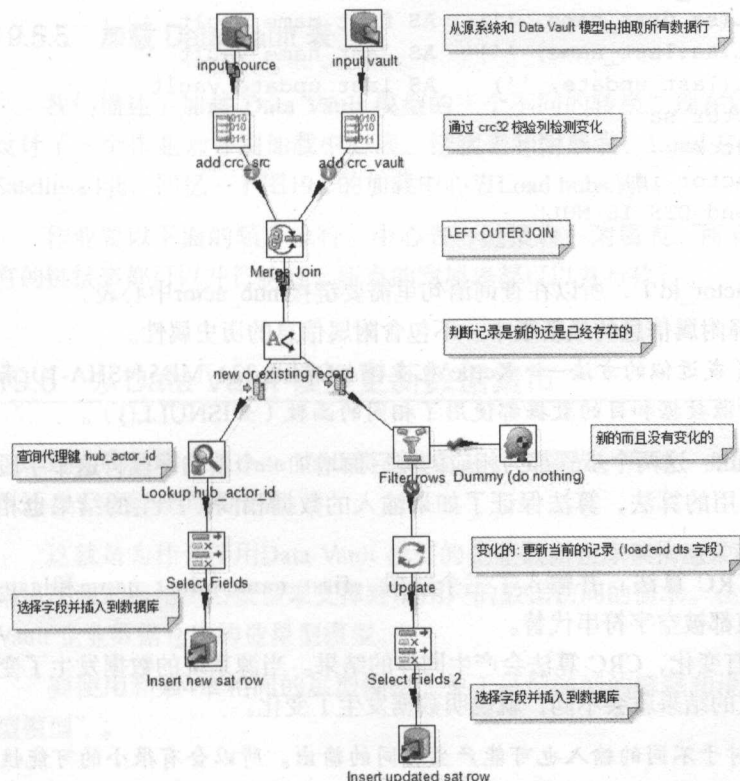


图19-5 并行加载所有中心表

附属表例子: *sat_actor*

加载附属表*sat_actor*的转换文件是*sat_actor.ktr*。转换如图19-6所示。

图19-6 *sat_actor* 转换

sat_actor 转换负责从源系统中的actor 表中采集actor的所有属性和属性的历史变化, 并以增量的方式加载到 Data Vault 模型的sat_actor 表中。

图19-6中主要步骤的描述如下。

- input source: input source步骤对sakila源数据库执行下面的SQL语句。

```
SELECT    actor_id
,         IFNULL(first_name, '') AS first_name
,         IFNULL(last_name, '') AS last_name
,         IFNULL(last_update, '') AS last_update
,         'sakila-db.actor' AS record_source
,         NOW() AS load_end_dts
FROM      actor
ORDER BY 1
```

该 SQL 语句从源数据库中选择所有将要插入到附属表中的actor的属性。在下一个步骤中, 对这些属性创建一个循环冗余校验 (CRC) 列来检查变化。因为一般 CRC功能不支持NULL值, 所以我们使用MySQL的ISNULL函数。其他数据库系统也都有类似函数。

load_end_dts字段需要的转换在后面使用, 这里我们直接使用 MySQL的NOW() 函数加载当前的时间戳。

- input vault: 类似于中心表例子和链接表例子, 从Data Vault 模型中抽取数据, 用于后面的Merge Join步骤。

```
SELECT    actor_id
,         sat_actor_id
,         hub_actor_id
,         IFNULL(sa.first_name, '') AS first_name_vault
,         IFNULL(sa.last_name, '') AS last_name_vault
,         IFNULL(last_update, '') AS last_update_vault
FROM      sat_actor sa
INNER JOIN hub_actor
USING     (hub_actor_id)
WHERE     load_end_DTS IS NULL
ORDER BY 1
```

因为你需要业务主键 (actor_id), 所以在查询语句里需要链接 hub_actor中心表。

WHERE 语句确保你选择附属信息的最新版本。不包含附属信息的历史属性。

注意: 当你使用CRC 技术 (或近似的方法——Kettle 也支持 ADLER 32、MD5和SHA-1) 来检测变化的数据, 确保你对源数据和目的数据都使用了相同的函数 (如ISNULL())。

- add_crc_src和add_crc_vault: 这两个步骤都为相应的流后面增加一个新的字段, 这个字段是计算字段, 它使用通用的算法, 算法保证了如果输入的数据相同, 产生的结果也相同。

在本例中, 我们使用 CRC 算法, 并输入了三个字段, first_name、last_name和last_update。所有的NULL 值都被空字符串代替。

只要源系统中的数据没有变化, CRC 算法会产生同样的结果。当源系统的数据发生了变化, 经过 CRC 算法产生的结果就会不同, 就说明数据发生了变化。

注意: 理论上, CRC 算法对于不同的输入也可能产生相同的输出。所以会有很小的可能性不能检测出源数据的变化, 我们这里忽略了这种可能性。如果想不忽略这种可能性, 就必

须要做字段对字段的比较，这样会使处理性能急剧降低。

- Merge Join: 类似于中心表的例子，Merge Join步骤对两个数据来源的数据的业务主键执行LEFT OUTER JOIN 操作。
- new or existing record: 下一个步骤判断是否正在处理一个新的或者已存在的记录。如果sat_actor_id的值是NULL，就一定是一个新记录，否则就是一个已存在的记录。新的记录会被送到Lookup hub_actor_id步骤，而已存在的记录会被送到Filter rows步骤。
- Lookup hub_actor_id: 类似于前面链接表的例子，这里需要把业务键（actor_id）转换为代理键（hub_actor_id）。
- Insert new sat row: 该步骤插入一个新行，新行包括下面几个字段，first_name、last_name、hub_actor_id、record_source和last_update。
另一个路线分支，是处理已存在的记录，Filter rows步骤是这个分支路线的起点。
- Filter rows: Filter rows步骤比较两个CRC32 计算结果。如果字段值都一样，这行就被忽略。如果字段值不一样，需要做下面两件事情。
 - 为当前记录设置结束时间。
 - 插入一条新的记录（开始时间是现在的时间）。
 这两件事情，就是下面两个步骤要做的。
- Update: 该步骤将附属表的load_end_date字段设置为数据流里的load_end_date字段（在input vault步骤中加入的这个字段）。
- Insert updated sat row: 最后往附属表里插入一行。行里包括first_name、last_name、hub_actor_id、record_source和last_update。

19.5.5 加载 Data Vault 表

我们描述了加载 Data Vault 模型的三个不同的转换，现在来看如何执行这些转换。我们设计了三个作业来分别加载中心表、链接表和附属表：Load Hubs.kjb、Load Links.kjb和Load Satellites.kjb。回忆一下图19-5的加载中心表Load hubs.kjb。

作业要以下面的顺序执行：中心表→链接表→附属表。所有的中心表都可以并行执行，所有的链接表都可以并行执行，所有的附属表都可以并行执行。

19.6 从Data Vault 模型更新数据集市

现在我们明白了Data Vault模型非常适合于集成、存储，并能保存有价值的信息。但是，它不适合于密集查询或报表。

这就是为什么使用Data Vault 模型的企业数据仓库架构通常都要包括一个数据集市层，数据集市层通常使用星型模型来支撑终端用户的数据访问的需求。在本节中，我们演示如何基于Data Vault 企业数据仓库构造星型模型。

要使用和第4章相同的星型模型。关于星型模型的解释和说明，参考第4章“租赁业务的星型模型”。

19.6.1 ETL解决方案例子

这里的ETL例子非常类似于第4章的例子（一些代码甚至是一样的）。但我们也会提供完整的例子，这样可以充分了解Data Vault 模型和它的特点。接下来的部分描述了将数据加载到星型模型的转换。大多数转换都是单独介绍的，除了dim_staff和dim_store，因为它们和dim_customer类似。

注意：dim_date和dim_time是静态维度，初始化时构建，以后不需要改变。将来这两个维度的转换文件分别是dim_date.ktr和dim_time.ktr，和第4章使用的完全一样。在本章不再讨论。

19.6.2 dim_actor 转换

dim_actor 表由dim_actor.ktr 转换加载数据，该转换见图19-7。

因为dim_actor 不使用缓慢变更维度（星型模型里只保存演员的最常使用的属性，历史属性不保留），加载dim_actor的转换非常简单。

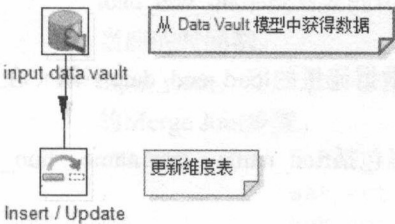


图19-7 dim_actor 转换

Table input类型步骤后面就是一个“Insert/Update”类型步骤，该步骤负责往数据库中插入新数据或更新已有数据。

转换里两个步骤的说明如下。

- Input Data Vault: 在本步骤中，下面的SQL查询把相应的数据加载到Data Vault 模型中。

```
SELECT      sa.last_update      AS actor_last_update
,           ha.actor_id         AS actor_id
,           sa.first_name       AS actor_first_name
,           sa.last_name        AS actor_last_name
FROM        sat_actor sa
INNER JOIN  hub_actor ha
USING      (hub_actor_id)
WHERE      sa.load_end_dts IS NULL
```

这个步骤从sat_actor 表中读取相关属性。因为要用到actor_id字段，所以SQL语句里还链接了hub_actor 表。因为我们不需要历史数据，所以用了WHERE sa.load_end_dts IS NULL 条件，只获取最新属性。

注意：尽管现在不需要演员的历史属性，将来可能会需要。如果将来需要，也可以从Data Vault 模型中获取，它保留了演员的全部历史属性的变化记录。

- Insert/Update: 感谢 Insert/Update步骤的功能，才使这个转换这么简单，才使你不用再自己做数据变更。


```
, DATE(sc.create_date) AS customer_created
, hc.customer_id AS customer_id
, sc.first_name AS customer_first_name
, sc.last_name AS customer_last_name
, sc.email AS customer_email
, sc.active AS customer_active
, sc.address AS customer_address
, sc.district AS customer_district
, sc.postal_code AS customer_postal_code
, sc.phone AS customer_phone_number
, sc.city AS customer_city
, sc.country AS customer_country
FROM
sat_customer sc
INNER JOIN
hub_customer hc
USING
(hub_customer_id)
ORDER BY
hc.customer_id, sc.load_dts
```

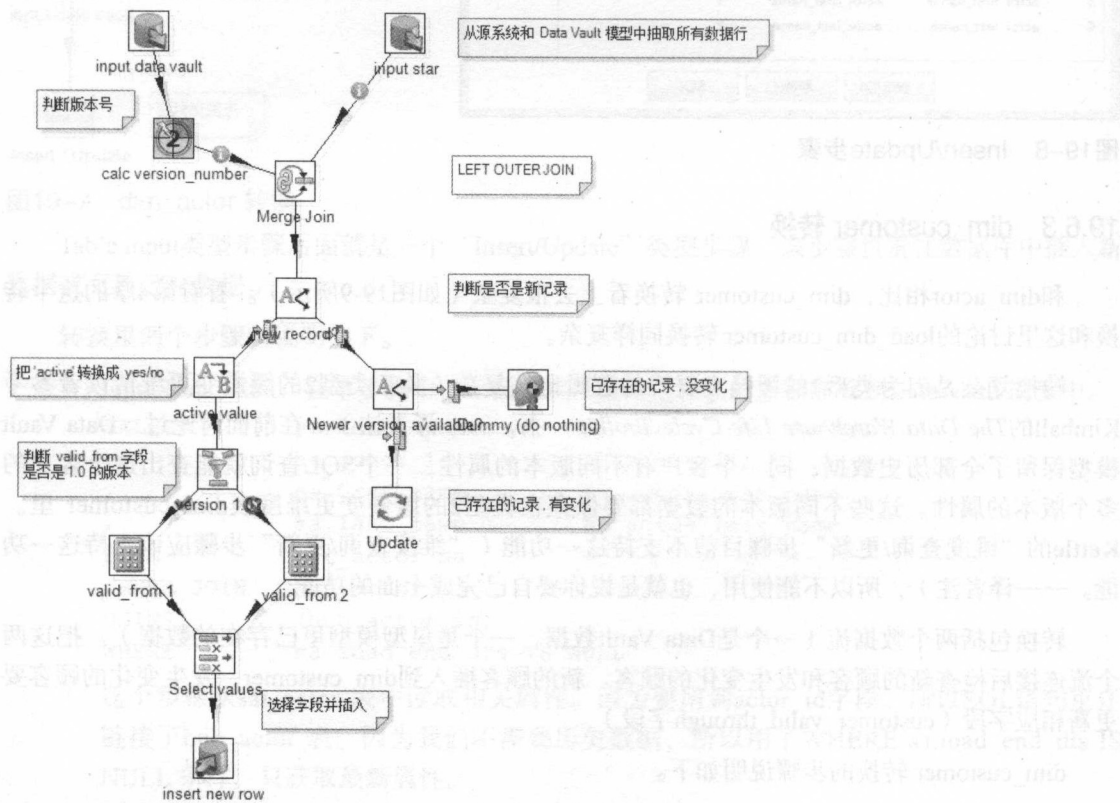


图19-9 dim_customer 转换

这个查询并不复杂。它从附属表sat_customer中查找相应字段，因为还需要customer_id字段，所以附属表还要连接到中心表hub_customer。这里需要注意 customer_valid_from和customer_valid_through字段，这两个字段是基于Data Vault 模型的load_dts和load_end_dts字段。

Data Vault 模型不知道customer 表里某个客户属性的第一个版本的开始生效时间。属性的第一个版本通常是在加载到Data Vault中之前就创建的。所以你要依赖源系统sakila 库的create_date字段。如果没有这个字段（例如dim_store和dim_staff 表），你要使用1970/01/01这样的日期。

在第4章中，我们介绍了如何把customer_valid_through 设置为2199/12/31。这里要使用同样的转换，通过IFNULL 函数来实现。

- input Star: input Star步骤执行了下面的SQL查询。

```
SELECT      customer_key
,           customer_id                AS star_customer_id
,           customer_valid_from        AS star_customer_valid_from
FROM        dim_customer
ORDER BY    customer_id, customer_valid_from
```

这个查询很简单，只是查询维度表里存在的列，不需要太多解释。

- calc version_number: 在两个流合并到一起前，Data Vault 流使用了一个技巧，就是“Add fields changing sequence”（分组序列——译者注）。这个步骤生成一个序列字段，每当分组字段的值发生变化，序列字段的值就重新计算（见图19-10）。这个序列字段就是我们要使用的版本号。如果以customer_id和load_dts字段排序，并把customer_id作为分组字段，序列字段里的版本号就会累加，如果customer_id 发生变化，版本号又从1 开始。

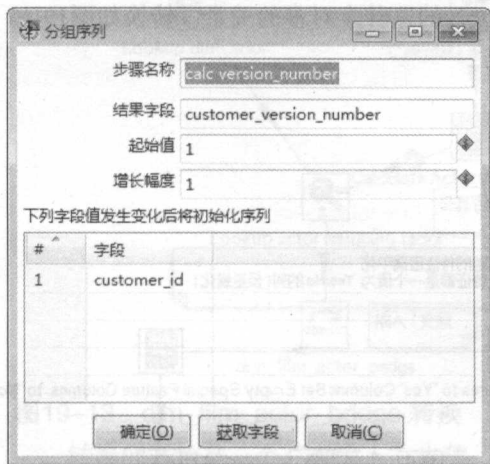


图19-10 计算版本号

接下来就是Merge Join步骤，该步骤使用customer_key字段把两个流连接起来。可以看出哪一行是新的，哪一行是已经存在的。已存在的行可能需要更新。

- Newer version available?: 当一个客户的某个属性发生了变化（如有一个新的地址），DataVault 模型的附属表里就会增加一行。这一行也要增加到星型模型的dim_customer 表中，另外当前行也要更新（更新customer_valid_through字段）。“Newer version available?” 字段用来检查是否发生变化。下一个步骤“Update”，用来更新customer_valid_through字段。
- Version 1.0?: 如前所述，某个客户的第一个版本属性的customer_valid_from字段需要一个特殊的值。这个值要被设置为create_date 而不是customer_valid_from字段。“Version 1.0?” 步骤过滤customer_version_number字段的值，把版本号是1的记录分流到下一个步骤。

图19-11 显示了该步骤的设置窗口。

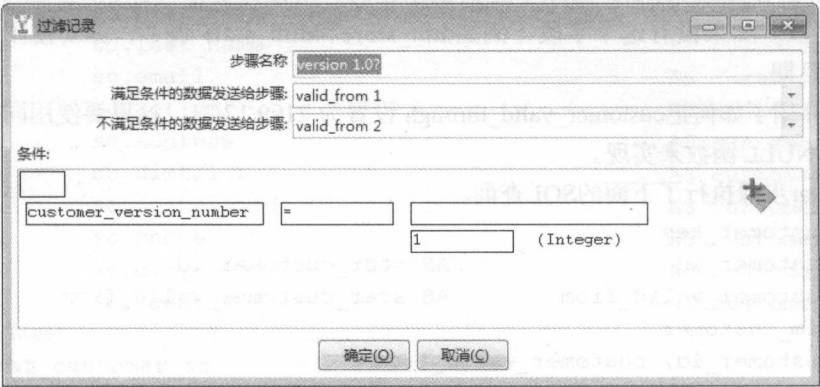


图19-11 customer_version_number

19.6.4 dim_film 转换

dim_film的转换很简单，如果不考虑[film_has ...]和[field_in_category...]这两组字段，dim_film转换类似于dim_actor 转换。[film_has ...]这组字段实际是把 special_features这个字段扁平化的结果。[field_in_category...]这组字段是基于 film和category的这种多对多的关系。转换如图19-12 所示。

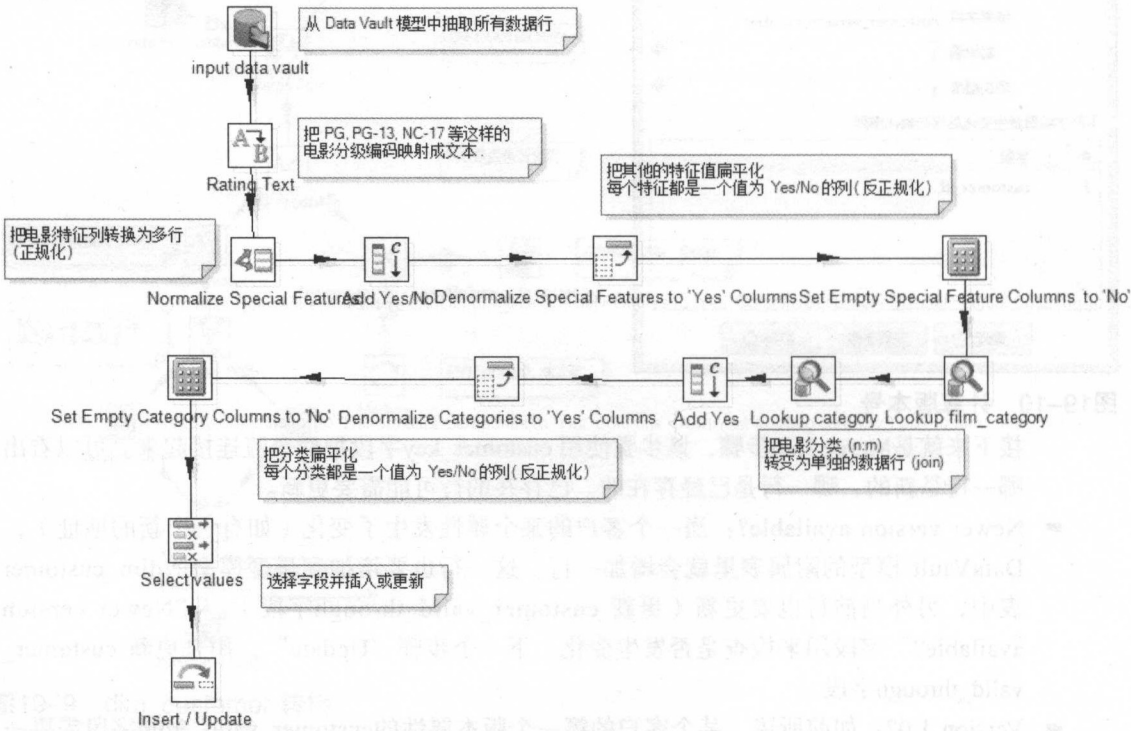


图19-12 dim_film 转换

尽管这个转换看上去比较复杂，我们也不在这里讨论了。这个转换非常类似于第4章已经介绍过的load_dim_film转换。

19.6.5 dim_film_actor_bridge 转换

电影和演员之间是多对多的关系（一个演员演过多部电影，一部电影里也有多个演员）。每个租赁业务（事实表的粒度）记录了一个电影光碟的租赁情况。就是说 dim_actor 维度表不会直接关联到 fact_rental 事实表。

因为我们想看到演员的数据，所以这里就要一个桥接表。这个桥接表在 dim_film 和 dim_actor 维度表之间构造了一种多对多的关系。

在第4章中转换 load_dim_film 已经包含了往 dim_film_actor_bridge 表中加载的逻辑。在本章，我们把这个逻辑分离出来，成为一个新的转换，如图19-13所示。

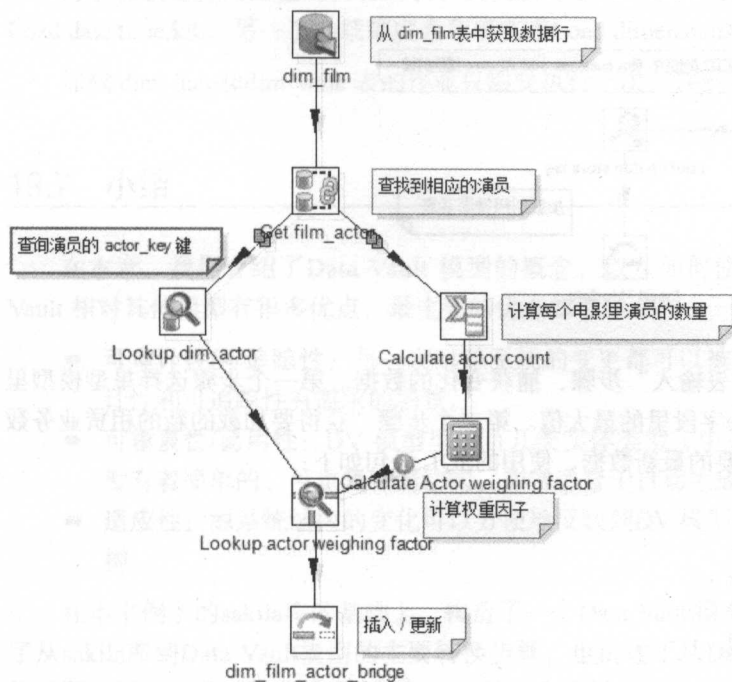


图19-13 dim_film_actor_bridge 转换

转换的开始是一个“表输入”步骤，这个步骤从 dim_film 读取主键值 film_id。下一个步骤是“数据库连接”类型的步骤，为每个 film_id 寻找对应的 actor_id。

结果被分成两个独立的流。左侧的流使用“数据库查询”类型步骤去匹配 actor_id，找到它对应的代理键 actor_key。右侧的流使用“分组”步骤计算每个电影里的演员个数，并在下一个“计算器”步骤计算演员的权重因子为：1/演员个数。最后，两个流在“查询演员权重因子”步骤中会合，使用合并后的结果更新 dim_film_actor_bridge 表。

19.6.6 fact_rental 转换

本章要介绍的最后一个转换是加载事实表 fact_rental 的转换。转换的名称是 fact_rental.ktr。这个转换非常类似于第4章的 load_fact_rentals 转换，只是前两个表输入步骤不同，我们下面详细介绍这两个步骤。转换如图19-14所示。

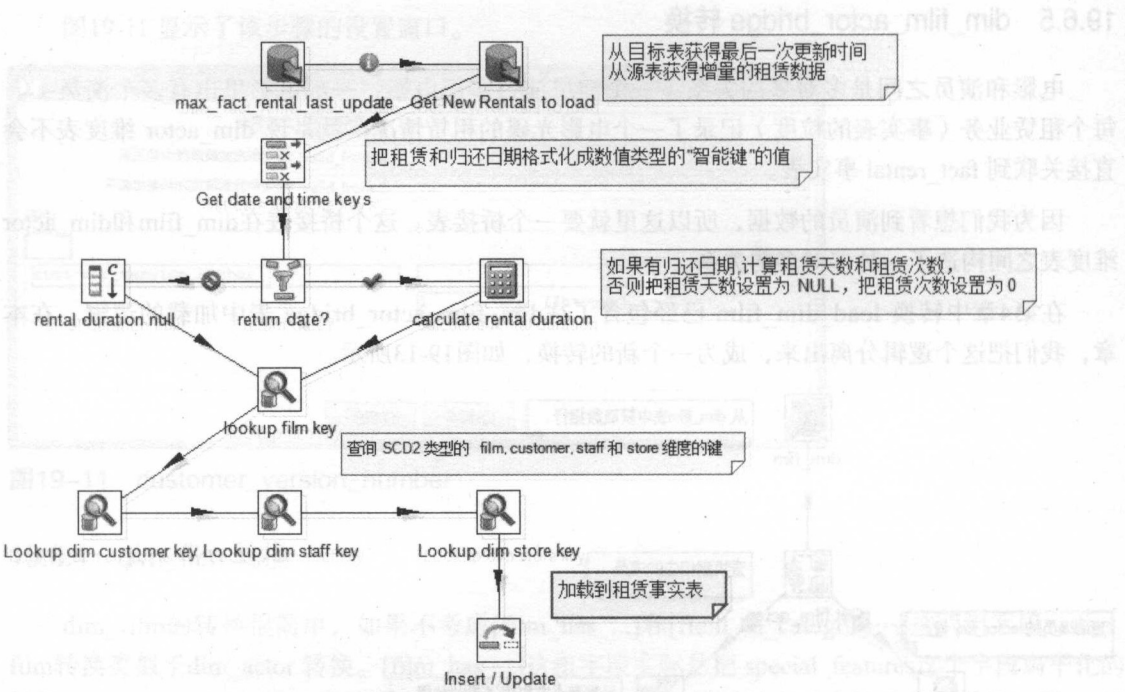


图19-14 fact_rental 转换

fact_rental的最开始是两个“表输入”步骤，捕获变化的数据。第一个步骤选择星型模型里fact_rental表里的rental_last_update字段里的最大值。第二个步骤“获得要加载的新的租赁业务数据”，从Data Vault模型里获得需要的最新数据，使用的SQL 语句如下：

```
SELECT      hr.rental_id
,           sr.rental_date
,           sr.last_update
,           sr.return_date
,           hc.customer_id
,           hs.staff_id
,           hstore.store_id
,           hf.film_id
FROM        link_rental lr
INNER JOIN  hub_rental hr
USING      (hub_rental_id)
INNER JOIN  sat_rental sr
USING      (hub_rental_id)
INNER JOIN  hub_inventory hi
USING      (hub_inventory_id)
INNER JOIN  hub_customer hc
USING      (hub_customer_id)
INNER JOIN  hub_staff hs
USING      (hub_staff_id)
INNER JOIN  link_inventory li
USING      (hub_inventory_id)
INNER JOIN  hub_store hstore
USING      (hub_store_id)
INNER JOIN  hub_film hf
```

```
USING      (hub_film_id)
WHERE      sr.last_update > ?
```

尽管这个SQL 看上去很复杂，其实并不难理解。因为要插入到 fact_rental 表里的数据来源于 Data Vault 模型里的多个表里的数据，包括租赁数据、客户数据、商店数据、员工数据和电影数据。这个查询也体现了Data Vault 模型的结构；不同的数据都存在于自己的表里，这样在添加和更新数据时比较灵活。查询只需要把所有包含所需数据的表连接起来即可。

19.6.7 加载星型模型里的所有表

为了加载星型模型里的所有的表，我们设计了两个Kettle作业，一个加载日期和时间维度，Load date time.kjb，另一个加载维度表和事实表Load dimensions facts.kjb。

加载 dim_date和dim_time 表的作业只需要执行一次，并不真正属于ETL过程。

19.7 小结

在本章，我们介绍了Data Vault 模型的概念，以及如何使用这个模型构建数据仓库。Data Vault 相对其他模型有很多优点，最主要的优点如下。

- 可审计性/可追踪性：每一次对源系统的变更都可以被捕获到，DV更适合于对数据的审计性和可追踪性有需求的场景。
- 可重复性/易用性：DV 模型里只有几类实体类型（中心表、链接表、附属表），DV 模型有着简单的、可重复性的结构，非常适合于自动生成模型和ETL代码。
- 适应性：源系统结构的变化可以方便地反映到DV 模型中，DV 模型不用修改现有的表结构。

在本书例子的sakila库的基础上，构造了一个Data Vault模型，作为演示用例。我们详细讲述了从sakila库到Data Vault模型的主要转换步骤，也讲述了从Data Vault模型到数据集市的主要转换步骤。通过本章，我们也了解到Kettle是一个通用的ETL工具，可以使用在任何数据仓库的场合。

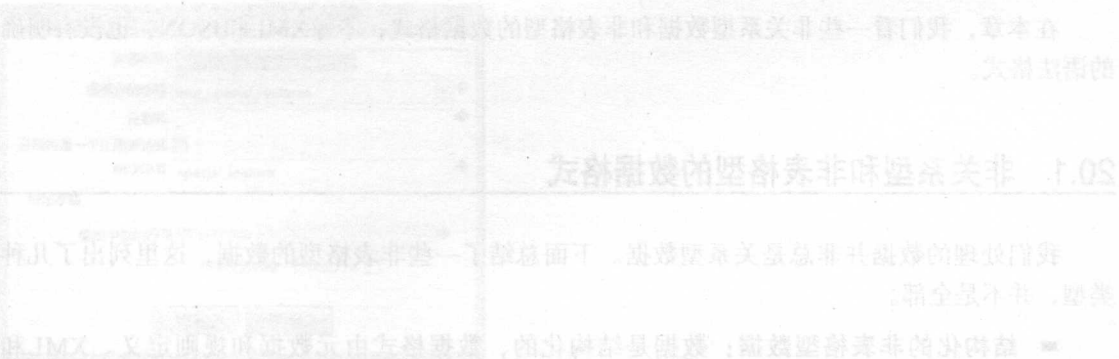


图20-1 “拆分分为多行”步骤的配置

第20章 处理复杂数据格式

在典型数据仓库场景下，ETL过程从业务系统抽取数据加载到数据仓库中。大多数情况下，业务系统和数据仓库都使用关系型数据库来存储。对于ETL过程来说，意味着数据都是可以通过关系型数据的格式来访问的。

通过第6~9章的内容，我们已经了解到：关系型的数据格式并不意味着转换会变得简单。但至少数据和源数据是分离的，数据之间的关系也是明确的：数据行把各个属性的值串起来，外键清楚地描述了数据行之间的关系。

或许更重要是关系型数据可以简化转换。无论数据格式怎样变化，输入和输出都是以表格的方式来表现的，也就是一组行的集合的方式，每行又可以明确地被分成若干列。

如果数据源不是表格的形式呢？在第21章，我们要读取XML和JSON格式的数据，它们允许任意层次的嵌套，它们是非表格的数据。尽管是非表格格式的数据，但这些数据的数据格式的语法清楚，也明确区分了数据和元数据。

在本章，我们看一些非关系型数据和非表格型的数据格式，不像XML和JSON，也没有明确的语法规式。

20.1 非关系型和非表格型的数据格式

我们处理的数据并非总是关系型数据。下面总结了一些非表格型的数据，这里列出了几种类型，并不是全部。

- **结构化的非表格型数据：**数据是结构化的，数据格式由元数据和规则定义。XML和JSON都属于这类。将在第21章详细讨论这类数据。

- **非结构化的表格型数据：**数据包括了一组数据行的集合，每个行也被分为了几个字段。但是，表里可能包含了有多个值的字段列或者有重复的一组字段。这种情况下，数据不是结构化的。
- **半结构化的和非结构化的数据：**在这种情况下，数据是以哪种规则组织到一起的，并不清楚，或者规则太复杂而不能规范化。
- **键值对：**数据的每一行包含一个键和一个值。

本章接下来要针对上面的后三种情况介绍几个例子，详细说明可以使用Kettle的哪个功能来处理这三种情况。

20.2 非结构化的表格型数据

当有下面两种情况之一时，表格数据就是非结构化的：

- 记录的某个字段里包含了多个值
- 重复的字段组

如果数据包含了上面两种格式之一，就需要使用ETL过程对这些数据进行重构，以把数据转换成关系型的数据格式。尽管数据不一定要以关系型格式保存在数据仓库中，但把数据转换成关系型格式，可以便于后面的清洗工作。

20.2.1 处理多值字段

多值字段是记录的一个字段里的值是由一组值组成的，而不是一个值。在第4章我们遇到过多值字段的例子：sakila库的film表里的special_features字段，这个字段使用了MySQL 特定的SET数据类型（见图4-1）。这种类型的数据是以逗号分隔的一个字符串来表示，我们称之为集合。

使用“列拆分为多行”步骤

在图4-16的load_dim_film 转换里，我们使用了“列拆分为多行”步骤把包含多值的字段拆分为多个行，每一行相应字段里只包括一个值。在load_dim_film 转换里，“列拆分为多行”步骤被命名为“Normalize special features”。第4章没有单独演示这个步骤，在这里详细讨论一下。图20-1 显示了这个步骤的配置。



图20-1 “列拆分为多行”步骤的配置

在这个对话框里设置一些必要的属性。

在对话框的“要拆分的字段”设置项里，使用下拉列表从接收到的数据流里选择一个多值字段。在本例中，选择 film.special_features 字段。在 film 的表结构定义中，这个字段的定义如下：

```
special_features SET('Trailers', 'Commentaries',  
                    'Deleted Scenes', 'Behind the Scenes')
```

这个定义说明 special_features 多值字段里的值可以是SET 关键字后面列出的取值范围内多个值的组合。

在对话框的“分隔符”设置项里，设置用来分隔多值字符串的分隔符。因为MySQL的SET 列使用逗号作为分隔符，所以在这里我么设置为逗号(,)。这样，这个步骤就会对逗号分隔的每个值都生成一个新行。例如，在“ACADEMY DINOSAUR”这个电影里，special_features 字段里的值是“Deleted Scenes, Behind the Scenes”，因为这个多值字段包含了两个值，所以这行记录会被输出为两行。

多值字段被分隔后，字段就规范化了。分隔后的值在输出流中保存在一个新的字段里。对话框的“新字段名”用来设置这个新字段的名字。

对话框里的“附加字段”栏可以设置一个用来保存行号的字段。选中“输出中包括行号”设置项，可以生成行号。在“行号字段”输入框中设置要保存行号字段的名字。最后，可以选中“对接收到的每一行重置行号”，可以对每一个输入行重新从1开始计数。

20.2.2 处理重复的字段组

重复字段组就是某类型的一个字段（或一组字段）在一个表里出现多次。例如sakila 库的 film 表有两个列都引用了language 表：language_id和original_language_id，这两个字段可能就是重复字段组。

注意：尽管我们认为 film 表里引用 language 表的这两列是重复字段组，但很多数据库开发设计人员并不这么认为。他们认为引用 language 表的这两列都有自己的语义，所以把这两列认为是重复列的理由并不充分。尽管这个例子不能太充分说明问题，在这里使用这个例子也是因为sakila 库里只有这一个重复字段组的情况。

使用行正规化（列转行）步骤

可以使用行正规化步骤（也称为列转行步骤——译者注）来解决这种重复字段值的问题。在Spoon 里，“列转行”步骤在左侧树状列表的“转换”类别下。sakila-normalize-repeating-groups 转换演示了列转行步骤。转换和列转行步骤的配置对话框如图20-2所示。

设置对话框的输入表中的“Key值”列用于设置区分字段类型的值。在本例中，重复字段组里两个字段，所以需要两个值来区分这两个字段。表里的每一行对应着重复字段组的一个字段。在输出流中增加了一个新的字段来保存类型值，这个新字段的名字在“Key字段”输入框里设置。

对每个输入行，有多少个Key值就会有多少输出行。Key值会保存在一个新的字段里，重复字段组将只保留一个字段，这个保留的字段在表格的“new field”列里设置，除了这个保留的字段外，其他字段都将被删除。



图20-2 行规范化（列转行）步骤使重复字段组规范化

20.3 半结构化和非结构化数据

在很多情况下，保存在文件里的数据，不规范，没有结构。如果人们要在这里找出需要的数据和数据之间的关系，完成自动化数据抽取，就需要定义一个描述数据格式的模式。这个模式如何定义就是数据抽取的前提条件：没有模式，就不可能做到自动抽取。

正则表达式是一种传统而又行之有效的方法，可以用来进行搜索和文本的模式匹配。正则表达式在很多语言中都被广泛支持，包括Perl、JavaScript、Java等。

注意：对正则表达式的详细讨论不在本书范围内。关于正则表达式有很多书籍和在线教程，学习正则表达式不会有什么困难。关于正则表达式可以参考下面几个网址：http://en.wikipedia.org/wiki/Regular_expression或<http://www.regular-expressions.info/>。

尽管写正则表达式有时比较困难，甚至非常困难，但在很多情况下都需要定义一个正则表达式来抽取数据。有时候，不值得花这么多的时间去写一个完全通配各种情况的正则表达式，这时候就要找一个不通配的，但足够好的正则表达式方案，放弃那些没有匹配上的数据。

有时候，写好的正则表达式不能匹配全部数据，这时就要总结那些没有匹配上的数据，以改进正则表达式，提高准确性。有时还要单独再开发一条转换路径，来处理这些未匹配上的数据。

注意：关于足够好的正则表达式方案，可以看一个E-mail地址的例子。关于这个例子在<http://www.regular-expressions.info/email.html>中有很详细的讨论。在这个例子里定义一个能匹配大部分E-mail地址（百分之九十九）并能检测大部分非法邮箱的正则表达式是比较简单的，用40个字符就可以完成：

```
^[A-Z0-9._%+-] +@ [A-Z0-9._]+\.[A-Z]{2,4}$
```

实际上，E-mail地址的规范在RFC 2822中定义，<http://tools.ietf.org/html/rfc2822#section-3.4.1>。和这个规范对应的完整的E-mail地址正则表达式有426个字符长，是上面简单正则表达式的10倍多。

为了更好地理解非结构化数据，我们看下面的movie.list文件的片段，movie.list文件由互联网电影数据库提供，这个数据库保存了大量的电影和电视剧的数据，网址是<http://www.imdb.com/>。

| | |
|---|-----------------------|
| #1 (2005) | 2005 |
| #1 Fan: A Darkomentary (2005) (V) | 2005 |
| \$100,000 Pyramid DVD Game, The (2006) (VG) | 2006 |
| \$50,000 Challenge, The (1989) (TV) | 1989 (unreleased) |
| 'Columbia' Winning the Cup (1901/I) | 1901 |
| 'Columbia' Winning the Cup (1901/II) | 1901 |
| 1 Second Film, The (2008) {{SUSPENDED}} | 2008 |
| 21 Days (1940) | 1940 (shot 1937) |
| Andrey Rublyov (1966) | 1966 (shot 1964-1965) |
| "#1 College Sports Show, The" (2004) | 2004-???? |
| "#1 Single" (2006) {Cats and Dogs (#1.4)} | ???? |
| "\$1,000,000 Chance of a Lifetime" (1986) | 1986-1987 |
| "\$10,000 Pyramid, The" (1973) | 1973-1988,1991-1992 |
| "\$10,000 Pyramid, The" (1973) {(1973-03-26)} | 1973 |
| "10 Years Younger" (2004/I) | 2004 |
| "10 Years Younger" (2004/I) {(#2.8)} | 2005 |

可以从 FTP 下载这个文件或其他类似文件，下载后是gzip的压缩文件，下载地址 <http://www.imdb.com/interfaces#plain>。

尽管这里的内容肯定可以通过正则表达式匹配，但规律并不明显。如果通过标准的文本文件输入步骤，最多只能读取每一行，而不能把每个字段解析出来。即使把整个一行作为一个字段，在以后的步骤中，还是要对一行数据进行解析。实际上，这段内容有下面一些规律：

- 每一行代表一部电影，电影可以是电影院播放的电影、电视播放的电影、电视连续剧、发行的视频、小型连续剧或视频游戏。
- 每行的开始都是一个主键，主键是电影名，用来唯一标识一部电影。
- 主键后面是一组不定长的制表符，接下来可能会有一对圆括号，括号内是电影类型的编码（TV是电视播放的电影，V是发行的视频，VG是视频游戏，mini是小型连续剧），然后又是一组不定长的制表符，后面是四位的发行年份。发行年后面可能还会有一组不定长的制表符和一对圆括号，来说明一些附加的信息，如 (shot 1937) 说明电影是在哪一个特定的年份拍摄的，(unreleased) 说明电影没有被公映。
- 如果没有明确的电影类型说明，电影类型是电影院放映的电影或者是电视连续剧。如果是电视连续剧，电影名字会用双引号引用起来。如果电影名前后没有双引号，就是电影院放映的电影。

Kettle 正则表达式的例子

为了从 IMDB电影文件里抽取数据，我们准备了imdb-movies这个转换。可以从本书网站下载第20章里的这个转换。除了转换，还要下载 movies.list.gz电影文件，电影文件从IMDB FTP下载，把电影文件放在和imdb-movies.ktr 转换同级的目录下。转换可以直接处理gz 压缩文件，所以不用解压。转换如图20-3所示。

这个转换的前面几个步骤就是本例的重点。转换后面的步骤主要是判断电影的类型。转换的最后步骤是把处理后的结果写到imdb-movies-output.txt文件中。处理后的结果就是我们需要的各个字段，主要有电影名、类型、章节号（对连续剧而言），等等。例如文本里的这一行：

"10 Years Younger" (2004/I) {(#2.8)}

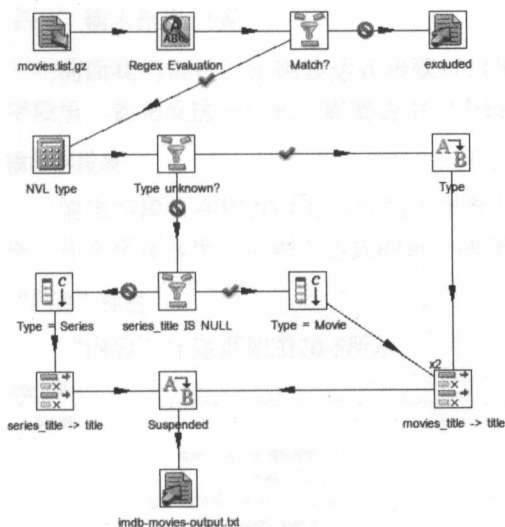


图20-3 使用正则表达式从IMDB电影文件中加载数据

经过处理就会变成：

```
line_number:566296
year:2004
part:I
secondary_title:(#2.8)
real_type:Series
title:10 Years Younger
```

注意上面只是抽取到的一部分字段的示例。实际的imdb-movies-output.txt文件里一行保存一条记录，字段之间都有分隔符，字段个数比上面的字段多。

配置正则表达式步骤

正则表达式步骤使用Java正则表达式语法，这个步骤在Spoon设计器的脚本类别下。关于正则表达式语法。见java.util.regex包里的Pattern类的API文档。这个文档在<http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html>。

正则表达式步骤有两个标签和一个输出字段的设置表格。下面分别介绍这两个标签。

“设置”标签

在图20-4中imdb-movies转换的正则表达式步骤的对话框里有一个“设置”标签（Settings）。在“正则表达式”列表框里需要输入一个正则表达式。这个正则表达式应该匹配完整的字段值，而不是字段值里的一部分。正则表达式可以包括Kettle变量，如果用了变量，要选中“使用变量替换”复选框。

对于imdb-movies转换来说，正则表达式非常复杂，要有多行。几乎每一行后面都有一个注释符（#）。表达式里的多行的结构、空白符和注释符都是可选的，只是为了使正则表达式看上去更易于理解。

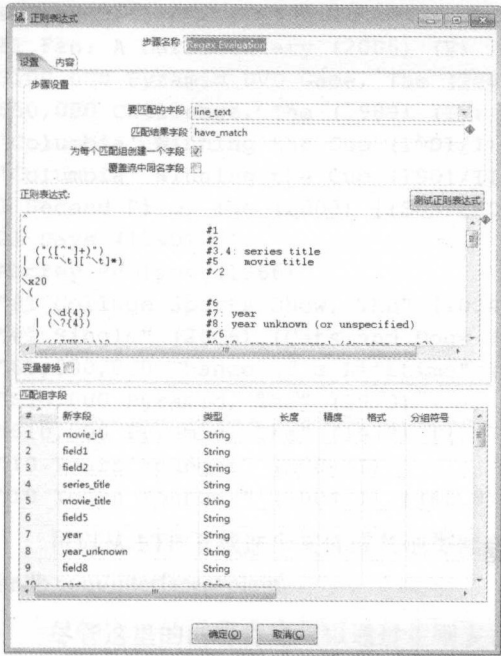


图20-4 正则表达式步骤的“设置”标签页面和输出字段表格

正则表达式步骤里，一些注释解释了要匹配的文本的类型，其他注释只标识出了字段组。
对于不熟悉正则表达式的读者，让我们看图20-4的正则表达式的一个片段。

```
(
    #2
    ("([^\"]+)")
    #3,4: series title
|
    ([^"\t][^\t]*)
    #5 movie title
)
#2
```

在上面片段的第一行，由左圆括号开始一个捕获组。捕获组是Kettle使用的机制，捕获组用来捕获正则表达式匹配到的数据，并把捕获到的数据转化为一个字段。上面的代码片段是第二个捕获组的开始，所以后面的注释是#2。第二个捕获组在片段的最后一行结束，最后一行后面的注释是#2。

下面一行是匹配标题的正则表达式。我们之前曾说过电视连续剧的标题以双引号作为标识符，这里的正则表达式也要匹配双引号：一个双引号后面跟着`[^\"]+`，这些字符表示非双引号的任何一个或多个字符，这些字符后面又跟着一个双引号。双引号里面的这些字符构成的正则表达式同样也使用一个圆括号括起来。这样就出现了圆括号的嵌套，有两个捕获组，在注释里说明这两组是#3,4。

下面一行是一个管道符`|`，这个符号表示或者的关系。管道符后面的正则表达式是第五组（#5），这个正则表达式的前半部分是`[^"\t]`，表示一个非双引号，非制表符的字符。这个正则表达式的后半部分是`[^\t]*`，表示非制表符的一个或多个字符。这个正则表达式是用来匹配电影名称的，从前面的格式说明里我们知道，电影名称是一个以非双引号、非制表符开头的任意字符串，这个字符串以制表符为结束标志。（以双引号开头的任意字符串是电视剧的名称）。

在“步骤设置”栏里，还要指定输入流里用来匹配正则表达式的字段，这个字段在“要匹配的字段”输入框里设置。匹配的结果会保存在输出流的一个新的字段里，结果是一个Boolean值，用来说明要匹配的字段是否可以和正则表达式匹配。这个输出字段的字段名在“匹配结果

字段”输入框里设置。

前面我们讲过，正则表达式步骤可以把根据正则表达式捕获的每组字符串保存在一个输出字段里。要完成这一功能，需要选中“为每个匹配组创建一个字段”复选框。

输出字段表

输出字段表在图20-4对话框的下半部分。如果选中了“为每个匹配组创建一个字段”复选框，你必须在这里为正则表达式的每个捕获组都设置一个输出字段。

“内容”标签

“内容”标签页如图20-5所示。

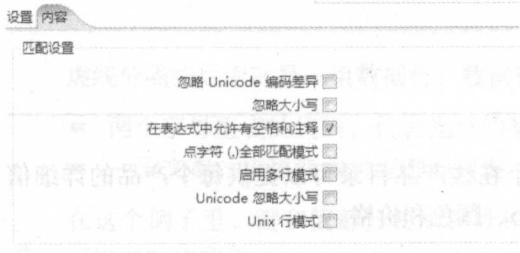


图20-5 正则表达式步骤的“内容”标签

在“内容”标签里是一些正则表达式匹配的设置项，根据实际需要，可以选中或取消这些配置项。

- “忽略Unicode 编码差异”：如果要匹配的字符串是ASCII编码而且想提高性能，可以选中这个选项。通常不要选中这个选项，除非你有充分的理由。
- 忽略大小写：默认情况下，选中这个选项可以在匹配ASCII 编码的字符串时忽略大小写。如果想对Unicode 编码的字符串忽略大小写，要选中“Unicode 忽略大小写”选项。
- 在表达式中允许有空格和注释：如果想让正则表达式跨多行并加注释，就要选中这个配置项。如果选中了这个选项，正则表达式里将不能包含空格，如果要匹配空格要使用\s或\x20表示方式。如果没有选中这个选项，正则表达式里的空格就是其字面本身含义，用来匹配字符串里的空格。这个选项打开或关闭，都会导致已有的正则表达式的匹配不正常，所以这个选项要在写正则表达式之前确定，而且一经确定就不能再修改。
- 点字符(.)全部匹配模式：通常点字符(.)可以匹配除了换行符的所有字符，选中这个选项后，点字符(.)可以匹配包括换行符的所有字符。
- 启用多行模式：通常情况下 ^和\$ 分别用来匹配一行的开始和结束。如果选中这个选项，^用来匹配每行的开始，\$用来匹配每行的结束。
- Unicode 忽略大小写：见上面的“忽略大小写”选项。
- UNIX行模式：通常情况下，回车/换行字符组合(\x13\x10)和换行字符(\x10)都可以识别成换行标志。如果选中了这个选项，只有\x10 被识别为换行标志，这个选项会影响到点字符(.)、^字符和\$字符的匹配结果。

默认情况下，这些选项都是未选状态。

校验匹配

无论正则表达式是否可以匹配输入字符串，正则表达式步骤都会根据输入产生一组输出结

果。所以你要检查正则表达式是否能够匹配上输入字符串。

在 imdb-movies 转换里，正则表达式步骤的后面使用了一个“过滤记录”类型的步骤来完成检查，步骤的名称是“Match?”。该步骤检查匹配结果字段，匹配结果字段的字段名在正则表达式步骤对话框里的“匹配结果字段”中设置。

如果匹配结果是true，数据行就沿着转换的剩余步骤继续执行，如果匹配结果是false，数据行就转到一个“文本文件输出”步骤，步骤名是“excluded”，所有未匹配上的数据行都保存在imdb-movies.excluded.txt。在生产环境下，这些文件至少要每天检查一次，以决定如何处理这些没有匹配上的数据。可能你会发现正则表达式还不够完善，这时就需要扩充正则表达式的逻辑以覆盖一些未匹配上的数据。

20.4 键/值对

键/值对是一种很普遍的数据格式。例如，一个在线产品目录可以提供每个产品的详细信息，在一个列表里列举了产品的所有属性，例如大小、颜色和价格。

见下面的例子：

```
LASERDISC LIST
=====
-----
OT:
LN: 3
LB: Philips
CN: 21317

LT: Stephen King's Nightmare Collection
PC: USA
CF: 16
CA: Movie
GR: Horror
LA: German
SU: -

RD: 1992
ST: Available
PR: DM 69.00

VS: PAL
CO: Color
SE: Digital
AL: -
AR: -
MF: Film
SZ: 12
SI: 2
DF: CLV
```


下面详细说明转换里的几个主要步骤。

文本文件输入

最开始，Imdb-laserdisc 转换使用和imdb-movies 转换同样的处理方式。文件里的内容是以逐行的方式读取的，这里的“文本文件输入”步骤不会把文件分成多个字段。它把整个文件的内容保存在一个字段里，字段名是line_text。

这个步骤还为输出流增加了一个line_number字段。从字段名我们就可以看出，这个字段用来保存行号，行号将来可以把所有属于同一行的数据连接起来。行号字段通过“文本文件输入”步骤的“内容”标签下的“在输出中增加行号”复选框来设置。

正则表达式

接下来，数据被送到了“Split key from value”正则表达式步骤。下面显示了正则表达式的部分代码：

```
^(
    (+)      # 1: record header
  | (
    (
      LN     # <LaserDisc Number>
    | LB     # <Label>
    ... many, many more lines of regex pattern code here...
    | AQ     # <Audio Quality>
  )         # /3
  : \x20?
  (.)?     #4: value
) #/2
$
```

正则表达式的顶层用来匹配记录头——虚线（捕获组1，保存在字段 header 里）或者用来匹配一个键/值对（捕获组2），键/值对又被进一步分割成键字段（捕获组3，字段key）和值字段（捕获组4，字段value）。在正则表达式步骤的后面，同样还使用了“过滤记录”步骤来验证匹配结果。

多行转换为记录

接下来要讨论的是“Java脚本”步骤，步骤名是“Mark Attribute rows with Id of header row”。JavaScript 代码如图20-7所示。

脚本里通过 if 语句检查header字段的值来判断当前行是记录头行还是键/值对行。在Java脚本步骤里，数据行里的字段可以直接作为变量使用，所以我们的代码可以直接这样写：

```
If (header) {
...
}
```

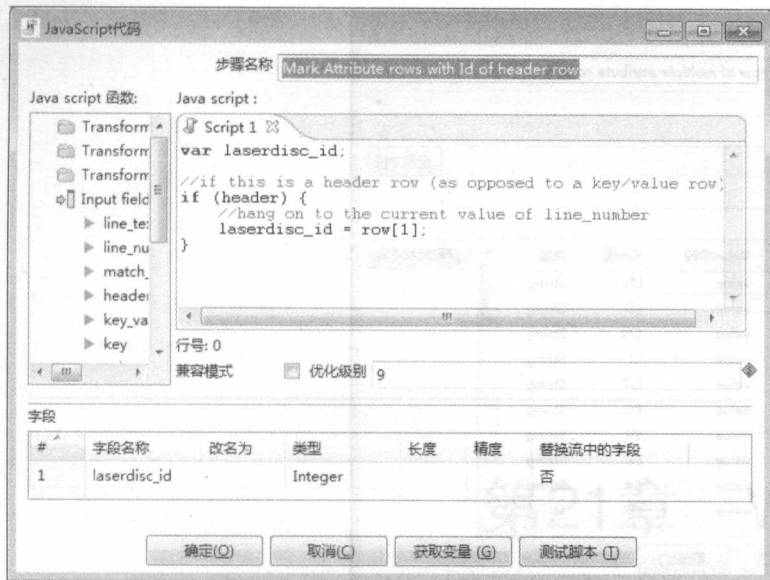


图20-7 “Mark Attribute rows with Id of header row” 步骤

如果是记录头行，header字段会包括一个虚线字符串，就是在正则表达式步骤捕获到的第一组#1。在Java脚本的if语句中如果表达式的值不是null，就会被认为是true，就会接着执行下面花括号里的代码。如果当前行是键/值对行，header字段就是null，花括号里的代码就会被跳过。

所以，如果当前行是记录头行，就会执行下面的代码：

```
laserdisc_id = row[1];
```

等号右边的row变量是内置变量，代表当前正在处理的数据行，它是一个字段数组。row[1]就代表了这行里索引1字段的值，因为下标索引是以0开始的，所以row[1]就是输入数据流里的第二个字段，就是在“文本文件输入”步骤里加入的line_number字段。

在图20-7下部的“字段”表格里，laserdisc_id被设置为输出流里的一个字段。通过设计器的“预览”功能可以看到从“Java脚本”步骤输出的数据流的格式，可以看到每一组记录的laserdisc_id都有相同的值，在下面的步骤中就要用到这个值。

反规范化：行转列

步骤名是“Make one row of multiple attribute rows”的步骤是一个“反规范化（行转列）”步骤。这个步骤会把输入的多行记录转换为多列的一行记录。该步骤的配置如图20-8所示。

“行转列”步骤把输入的数据按照事先设置的“分组字段”进行分组，这里的“分组字段”就是在Java脚本步骤里添加的laserdisc_id字段。分组后，所有laserdisc_id字段值相同的记录最终作为一条记录送到输出流中。

注意：“行转列”步骤要求分组字段 laserdisc_id 事先排好序，在这个例子里，laserdisc_id 字段是已经排好序的，所以不用再排序。如果要分组的字段事先没有排好序，需要使用“排序”步骤对分组字段排序后，再使用“行转列”步骤。

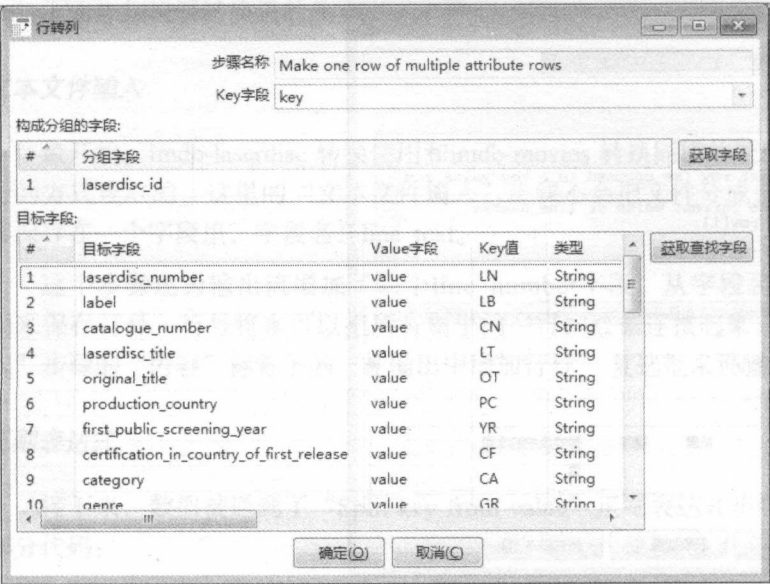


图20-8 “行转列”步骤

在对话框下部的“目标字段”表格里定义了要在输出流中增加的字段。要增加字段的字段名在表格里的“目标字段”列中设置。Kettle 使用下面的流程为输出流中添加这些字段：

- 对于分组内的每一行数据，Kettle 先获得关键字段的值，关键字段的字段名在“Key字段”输入框里设置。
- 在“目标字段”表格中的“Key值”一列中查找对应的关键字段的值。
- 如果找到了关键字段的值，Kettle 从“Value字段”指定的字段名中获得一个数据字段的值。
- Kettle 将该数据字段的值保存在输出流的一个新字段中，新字段的字段名在“目标字段”表格的“目标字段”列中设置。

20.5 小结

本章看了几个不符合关系型数据格式的例子，以及如何处理这些格式的数据。本章主要讲了下面几点内容：

- 几种非关系型数据的分类：一个字段里包含多个值的非关系型表格数据，包含重复字段组的非关系型表格数据；使用复杂正则表达式匹配的半结构化/非结构化数据；键/值对。
- 使用“列拆分为多行”步骤来正规化多值字段。
- 使用“列转行”步骤正规化重复的字段组。
- 使用“正则表达式”步骤抽取匹配正则表达式的数据，使用正则表达式的捕获组把捕获的数据分成不同的字段。
- 如何把属于同一条记录的键/值对作为一组。
- 使用“行转列”步骤把属于同一组的键/值对合并为一行。

第21章 Web Services

在第6章我们讨论了几种简单的Web数据抽取方式。本章我们深入了解Kettle 抽取Web数据和Web Services的功能，如何处理通用的数据交换格式。

在深入到细节之前，在本章的前面先解释为什么Web Services适合于ETL和数据整合工作，以及涉及的概念和技术。接下来本章介绍了几种Web方式交换数据的通用格式。本章的剩余部分说明用 Kettle 处理Web Services数据的一些典型场景。

21.1 Web 页面和Web Services

我们通常说的Web一般就是指Web页面。Web页面实际就是按照HTML规范编码的用来给人看的文件，通过HTTP 协议获得。通过Web 浏览器，用户就可以连接到服务器，服务器就是互联网的一部分。Web页面是超文本格式的，供人阅读的文档，文档里包含着链接到其他Web 页面的超链接。

HTTP 协议定义了客户端发出的请求如果通过互联网最终传到主机，主机如何发送一个包含有资源的响应。HTTP不只局限在超文本文件的传送上，只要数据编码正确，HTTP可以传送任何种类的数据。人们通常可能认为通过HTTP 传送的是静态文本或文档。实际并不一定如此，在大多数情况下也不是这样：响应数据一般是根据HTTP 请求自动生成的。

类似的，统一资源定位符（URL）也通过一个地址模式来唯一标识一个特定资源。尽管URL可以被看成是静态资源的地址，实际上URL 非常灵活，可以携带很多信息。URL可以用来发送一条命令，服务器接到请求后，可以执行这个命令（而不仅仅是抽取特定文档）。在这种情况下，返回的响应实际是命令执行结果的状态信息。

Web Services和Web页面不同,因为它们返回的信息往往是给机器处理的,而不是给人阅读的:它们返回的信息不适合人直接阅读。实际上,Web Services是不同应用程序之间数据交换的机制,它更像一个应用程序接口(API)。例如Netflix,这是一个Web Services的例子,位于<http://developer.netflix.com/docs>,或者下面的Web Services列表的例子:<http://www.webservices.net/WS/wscatlist.aspx>。还有一个非常全的Web Services的例子:<http://www.programmableweb.com>。

Web Services一般不使用HTML,使用更适合数据交换的格式,例如XML或JSON。在本章后面将介绍常用的数据交换格式。

Kettle Web 功能

Kettle 提供了一些处理Web数据的功能,将在后面讨论这些功能。本章将结合具体的实例详细描述这些功能。

一般HTTP步骤

Kettle 提供了两个通用的HTTP步骤:“HTTP Client”和“HTTP Post”。这两个步骤都位于设计器左侧步骤树的“查询”类别下。这两个步骤都完成HTTP请求,并把返回结果添加到输出流里。这两个步骤都可以直接设置URL,或者从前面的数据流中获取URL,并且这两个步骤都可以把输入流里的字段映射成URL里的查询参数。

这两个步骤的区别在于“HTTP Client”步骤发送HTTP GET请求,而“HTTP Post”步骤发送HTTP POST请求。另外使用“HTTP Post”步骤,你还可以设置HTTP请求报文里的消息体。例如,除了可以设置URL的查询部分的参数,可以通过POST的方式把参数作为消息体里的内容提交,或者把整个文件都作为消息体提交。

本章后面的例子里将描述这些步骤的细节。

简单对象访问协议

Kettle的“Web Services查询”步骤提供对简单对象访问协议(SOAP)的Web Services的支持。和一般的HTTP步骤类似,这个步骤也在设计器左侧树的“查询”类别下。该步骤用来查询基于SOAP的Web Services。

“Web Services查询”步骤使用某个SOAP服务提供的WDSL文档来判断当前的Web Services服务支持哪些操作。“Web Services查询”步骤将使用这些信息把输入流里的字段映射为请求里的参数,并把返回值加入到输出流里。

我们将在本章后面的一个SOAP的例子里详细讨论“Web Services查询”步骤。

RSS

Kettle 支持RSS输入和RSS输出。RSS在数据源定时更新的应用场景中用得越来越多。一般用于新闻和博客的更新,也用于发布股票报价和汇率。RSS也在本章后面的章节讨论。

Apache虚拟文件系统集成

Kettle也支持Apache的虚拟文件系统。这样可以在任何使用文件名的地方都使用URL。包括一些输入步骤里的文件名输入框，以及“转换”作业项里的转换文件名，“作业”作业项里的作业文件名。在Spoon里，也可以通过URL打开一个转换（主菜单→文件→从URL打开），或者把转换保存在VFS（主菜单→文件→另存为(VFS)）。

因为URL里的模式部分指明了URL使用的协议，所以URL不仅可以通过HTTP方式访问Web文件，还可以通过FTP方式访问.tar、.jar、.zip、.gzip等压缩文档，等等。

注意：关于Apache VFS可参考 <http://commons.apache.org/vfs>。关于URI语法和支持的文件参考 <http://commons.apache.org/vfs/filesystems.html>。

21.2 数据格式

尽管HTTP协议要求数据编码，但它不强制要求被传输数据的数据类型和格式。有一些格式在实际应用中广泛使用。我们将特别关注XML、HTML和JSON格式。

21.2.1 XML

XML是Web Services使用最广泛的格式，普遍用于互联网和数据交换。我们假设读者一般都熟悉XML格式和相关技术，如XPath和XML模式。

注意：关于XML资源，参考David Hunter编写的*Beginning XML*，Wrox出版社，2007。

Kettle提供了几个能处理XML数据的步骤和作业项。我们会在随后的例子里演示这几个步骤和作业项。对于不熟悉XML和相关技术的读者，我们快速介绍一下相关技术，使你对Kettle提供的XML处理功能有大致了解。

注意：除了对XML支持的通常功能，Kettle对一些特定的XML应用也提供支持，如RSS和SOAP。

Kettle里用来处理XML的步骤

Kettle提供了下面几个步骤来处理XML格式数据：

- “XML文件输入”步骤，位于设计器左侧步骤树的“输入”类别下。该步骤从一个文件、URL或输入流的一个字段里读取XML格式的数据。使用XPath表达式从XML文档中抽取节点和属性，然后转换为字段和记录并放入输出流中。在本章后面通过一个例子详细介绍该步骤。
- “XML输出”步骤，位于“输出”类别下。该步骤将接收到的数据流转换成一个或多个XML格式的数据，并把它保存为文件。
- “添加XML列”步骤，位于“转换”类别下。用于从输入流的数据构造XML字符串并作为一个新列。该步骤通常用于构造多层次嵌套的复杂XML。本章后面的生成XML文档的例子里详细讨论这个步骤。
- “XSL转换”步骤，也位于“转换”类别下。该步骤使用一个XSLT样式文件转换一个

从上一个步骤获得的XML文档，转换结果保存在输出流中。XSLT 语法不在本书的范围内。简单说，XSLT是处理XML数据的一个强大工具，可以用于抽取和生成复杂XML文件。凡是Kettle 其他 XML 类步骤不容易完成的需求，都可以考虑使用这个步骤。

- “XML 连接”步骤，位于“连接”类别下。这个步骤用来把一个XML 片段合并到一个XML文档中。用于生成复杂的、嵌套的XML 结构。本章后面的生成XML文档的例子详细讨论这个步骤。
- “使用 XSD 检验XML文件”步骤，位于“检验”类别下。该步骤可以使用一个XML Schema文件来检验一个 XML文件是否有效。XML Schema文件（也称为 XML 模式文件）是一种定义XML文件结构和数据类型的通用方法。通常用于定义Web Services服务的输入和输出的XML数据格式。XML Schema的语法也不在本书的讨论范围内，但我们将在本章后面的抽取数据生成 XML文档的例子详细讨论这个步骤。

注意：除了前面列举的这些转换步骤，目前的版本还有一些不推荐的XML 输入步骤：“XML 输入”和“XML 流输入”步骤，这些步骤目前位于“不推荐的”类别下。这些步骤可能会在Kettle的将来版本中移除，当构造新的转换时，尽量不要使用这些步骤。

Kettle XML 作业项

Kettle 也提供了一些处理XML的作业项，都位于“XML”类别下：

- “检查XML文件格式”作业项用来批量检查目录下的所有XML文件的格式是否正确。
- “DTD 验证”作业项通过DTD的方式验证XML文件的格式是否正确。
- “XSD 验证”作业项通过XML Scheme的方式验证XML文件的格式是否正确，和转换里的“使用 XSD 检验XML文件”步骤类似。在本章后面的生成XML文档的例子将详细描述该作业项。
- “XSLT 转换”作业项和转换里的同名步骤功能相同。

ECMAScript FOR XML

Kettle 除了提供上面处理 XML格式数据的步骤和作业项外，Kettle的JavaScript步骤还扩展了一种专用于处理XML的脚本语言，ECMAScript for XML，也称为E4X。

E4X 标准由ECMA维护，可以从这里下载：<http://www.ecma-international.org/publications/standards/Ecma-357.htm>。

21.2.2 HTML

HTML的全称是超文本标记语言。HTML 使用和XML 同样的语法结构：节点、属性和文本。除此之外没有其他相似点。

XML的目的是数据交换，HTML的目的是通过浏览器显示给用户阅读。为了达到这个目的，HTML 定义了一组有限的、固定的节点和属性，来定义文本的结构（如头部、段落、章节）或样式（如字体、颜色）。

尽管HTML 并不用来进行数据交换，但很多我们感兴趣的数据都是以 HTML格式提供的。尽管现在越来越多的公司或组织都在以Web Services的方式提供这些数据，但还有很多公司或组织不能或者不愿意这么做。在这种情况下，只能从网站上抓取这些页面。

不幸的是, Kettle并没有提供从 HTML页面中抽取信息的通用步骤, 但可以使用Kettle下载页面, 再利用像JavaScript、公式以及用户自定义 Java 表达式和类这样的步骤, 通过字符串的操作, 从 HTML中抽取需要的数据。

21.2.3 JavaScript Object Notation

JavaScript Object Notation (JSON, 发音Jason) 是目前Web Services应用中越来越多的一种数据交换标准, 甚至超过XML格式。JSON格式最初由Douglas Crockford在<http://www.ietf.org/rfc/rfc4627.txt>中提出。关于JSON 更多的介绍请参考<http://www.json.org/>。

XML和JSON格式的比较, 以及它们的优点和缺点, 不在本书讨论的范围之内。实际上, 两者最大的区别在于, XML是一种文档语言, 而JSON是一种对象串行化的格式。尽管这两种格式都便于机器识别, JSON 在目前流行的编程语言中更便于映射成对应的数据类型。

语法

从语法上, JSON是JavaScript 语言的子集。就是说从JavaScript 里可以直接访问和遍历JSON数据结构。下面是使用JSON来表示书的集合的一个例子:

```
[
  {
    "title": "Pentaho Solutions",
    "publisher": "Wiley",
    "isbn": "978-0-470-48432-6",
    "price": "50.00 USD",
    "pages": 658,
    "inPrint": true,
    "authors": [
      {
        "firstName": "Roland",
        "lastName": "Bouman"
      },
      {
        "firstName": "Jos",
        "lastName": "van Dongen"
      }
    ]
  },
  {
    "title": "Pentaho Kettle Solutions",
    "publisher": "Wiley",
    "isbn": "978-0-470-63517-9",
    "price": "50.00 USD",
    "pages": 744,
    "inPrint": false,
    "authors": [
      {
        "firstName": "Matt",
        "lastName": "Casters"
```

```

    },
    {
      "firstName": "Roland",
      "lastName": "Bouman"
    },
    {
      "firstName": "Jos",
      "lastName": "van Dongen"
    }
  ]
}

```

例子里演示了JSON的几个主要特点:

- 最外层的结构是一个数组, 数组里每一个元素是一本书。数组的标志是一对方括号: [和]。数组里的元素使用逗号分隔。
- 每本书使用对象的字面值来定义。对象字面值通过一组花括号来表示: {和}。
- 花括号里是对象的成员(属性)。一个成员是一组键/值对, 键和值之间使用冒号分割。键/值对之间使用逗号分隔。
- 在每个对象成员里, 冒号前面的键必须使用双引号作为标识符, 冒号后面是值。
- 成员的值必须是有效的JSON值。JSON值有五种: 字符串、数值、对象、数值和布尔值。
- 字符串类型的值也使用双引号作为标识符。在这个例子里, 第一本书 title成员的值是字符串 Pentaho Solution。
- 数字类型的值可以是整型或浮点型, 或者是指数表达式。在上面的例子里, 第二本书的页数是整型值744。
- 布尔值只有两个值, true或者false。在上面的例子里, inPrint对象的值是布尔类型值。

注意: 尽管JSON和JavaScript 语言紧密相关, 但 JSON在其他语言环境中也广泛使用, 在很多语言中都有开源的JSON 实现, 包括 C/C++、C#、.NET、Java等。在json.org 站点上有适合多种开发语言的JSON库的列表。

JSON 通常被认为比 XML 简单得多。如果只考虑这两种数据格式的规范, 这种看法是正确的。XML 语法由89 条语法规则定义。JSON的规范只有15条。最基本的JSON 规范只有6条。剩下的9条规范与字符串类型(3条)和数值类型(6条)的词法定义相关。

JSON, Kettle和ETL/DI

目前 Kettle 不提供生成和读取JSON数据的特性。但因为JSON 实际是JavaScript, 可以使用Java脚本步骤来访问JSON数据。将在本章的JSON例子里讨论这种方法。另一种方法就是你自己写插件。(在Kettle 4.1 及以后版本中提供了JSON 输入和输出步骤。——译者注)

除了最初的抽取和访问数据的方法, 处理JSON的转换也面临着和处理XML的转换同样的问题。这两种格式都被用于marshalling对象。嵌套式的数据结构常用于表达对象和对象之间的关系, 处理好这种数据结构, 才能设计好转换。

21.3 XML例子

在本节，将详细讨论 Kettle 处理 XML 数据的能力。因为 XML 是一种非常灵活的格式，可以用来表达很多种数据结构，所以不可能提供一个通用的例子。但是在 XML 底层，只包括了有限的几种结构，很多高层的数据结构都基于这几种基本结构。

在本节，我们先看一个包括演员和视频信息的 XML 文档。然后再看如何从这个 XML 文档里抽取数据，并把数据保存在第4章介绍过的 sakila 数据库中。最后，我们再看一个如何从 sakila 数据库中抽取数据并保存成 XML 文件的例子。

21.3.1 XML 例子文件

在 XML 的开发环境 Stylus Studio 中，有一个样本文件 videos.xml。可以从 <http://www.stylusstudio.com/examples/videos.xml> 下载这个文件。除了这个 XML 文件，还有一个描述这个 XML 文件结构的 XML Schema 文件。可以从 <http://www.stylusstudio.com/examples/videos.vsd> 下载这个 Schema 文件。

XML 例子文件结构

下面是 videos.xml 文件的代码片段。

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <actors>
    <actor id="00000015">Anderson, Jeff</actor>
    ...more actor elements here...
    <actor id="916503210">Sharif, Omar</actor>
  </actors>
  <videos>
    <video id="id1235AA0">
      <title>The Fugitive</title>
      <genre>action</genre>
      <rating>PG-13</rating>
      <summary>...text...</summary>
      <details>...more detailed text...</details>
      <year>1997</year>
      <director>Andrew Davis</director>
      <studio>Warner</studio>
      <user_rating>4</user_rating>
      <runtime>110</runtime>
      <actorRef>00000003</actorRef>
      <actorRef>00000006</actorRef>
      <vhs>13.99</vhs>
      <vhs_stock>206</vhs_stock>
      <dvd>14.99</dvd>
      <dvd_stock>125</dvd_stock>
      <beta>1.03</beta>
      <beta_stock>12</beta_stock>
      <LaserDisk>12.00</LaserDisk>
      <LaserDisk_stock>10</LaserDisk_stock>
```

```
</video>
...more video elements here...
<video>
...
</video>
</videos>
</result>
```

<result>文档节点下面包括了一个<actors>和一个 <videos>节点。这两个节点分别又包含一组<actor>节点和一组<video>节点。

<actor>节点有一个唯一的id属性，用来唯一标识文档内的<actor>。<actor>节点内包含的是一段文本，文本前半部分是演员的姓，后面跟着逗号，再后面是演员的名字。

<video>节点里包含了video 里的一组属性，如<title>、<genre>和<rating>。

<video>节点里还包含了一个或一组<actorRef>节点，每个<actorRef>节点里包括了某个<actor>节点的id属性值，说明视频里包含哪些演员。

映射到sakila例子库

在这个例子里，我们假设数据库表里的字段和videos.xml 里的节点有明确的映射关系。

- <actors>节点对应actor 表，每个<actor>节点是表里的一条记录。
- <videos>节点对应film 表，每个<video>节点是表里的一条记录。
- <video>节点内的<genre>节点对应 film_category 表里的一行。
- <video>节点内的<actorRef>节点对应 film_actor 表里的一行。

XML 结构和sakila数据库的映射关系如图21-1所示。

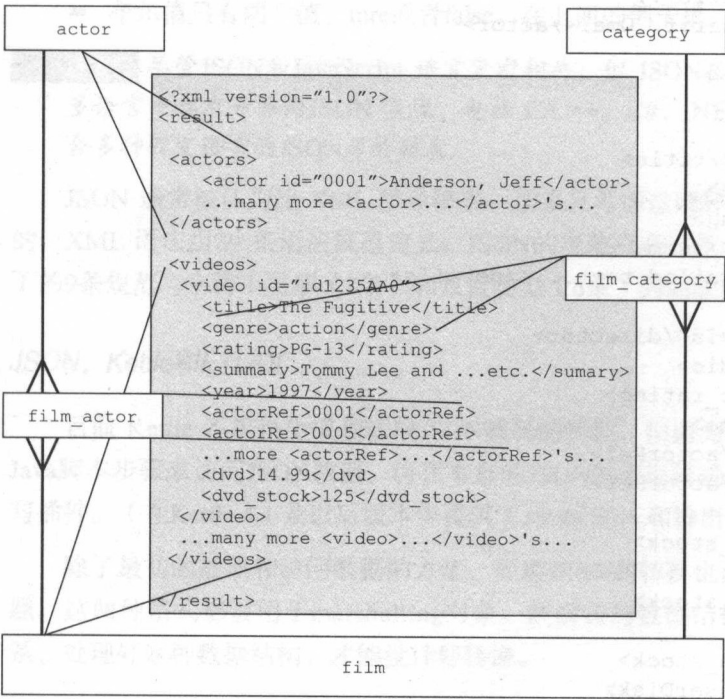


图21-1 videos.xml和sakila数据库的映射关系

21.3.2 从XML中抽取数据

为了从 videos.xml文件中抽取数据并加装到数据库中，我们设计了import_xml_into_db 转换。转换如图21-2所示。

注意：转换文件 import_xml_into_db.ktr 在本书站点（www.wiley.com/go/kettlesolutions）对应的目录下。另外要运行这个转换，需要先创建好sakila数据库和用户账户。如何创建数据库，参考第4章。

这个转换还依赖一个名称为 videos.xsd的XML Schema文件。这个文件需要和转换文件在同一个目录下。这个XML Schema文件也可以从上面提到的Stylus Studio 网站上下载。

最后，这个转换还要从Stylus Studio 网站上下载 videos.xml文件。所以在执行转换时要保证可以连接到网络。

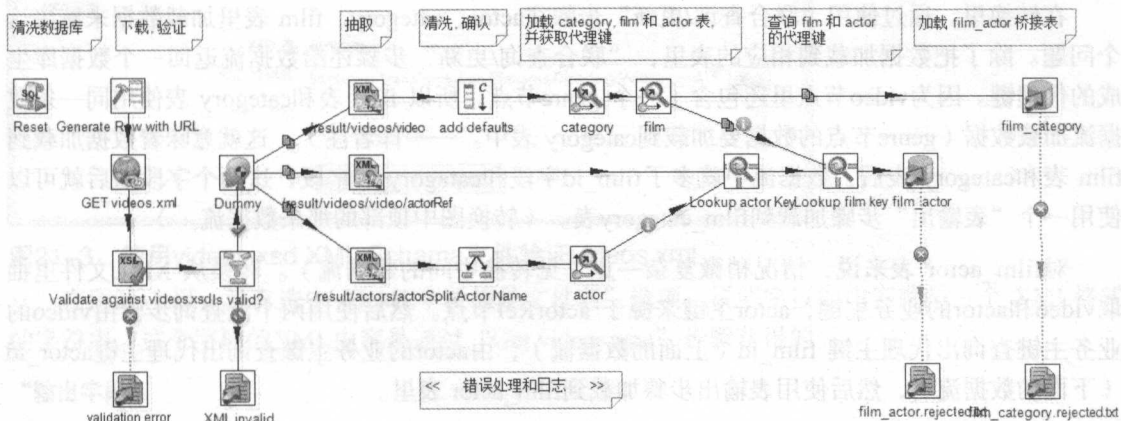


图21-2 import_xml_into_db 转换

import_xml_into_db 转换的总体设计

在我们深入了解 import_xml_into_db 转换细节之前，先了解转换中数据流的总体设计。在图21-2的上方，可以看到几个注释，每个注释解释了注释下面几个步骤的作用。

第一个要执行的步骤是一个“执行 SQL脚本”步骤，步骤名是Reset。在图21-2中，这个步骤位于左上角。该步骤在转换的初始化阶段执行，并没有参与到实际的转换中。该步骤的目的就是移去sakila数据库中标准数据集之外的任何数据。使转换有一个干净的数据环境。

初始化阶段后，转换运行的第一个步骤是“生成行”步骤，步骤名是“生成URL数据行”，该步骤也位于图21-2的左上角。“生成行”步骤生成一个字符串常量值，字符串常量值就是要抽取的videos.xml文件的URL。

生成的URL被送到一个“HTTP client”步骤，步骤名是“GET videos.xml”，该步骤用来抽取videos.xml文件。然后videos.xml文件的内容也保存到了输出流的一个字段里。“HTTP client”步骤的配置在本章后面将详细讨论。

下载了XML文件后，使用“Validate against videos.xsd”步骤验证该XML文件是否是转换要求的格式。这个验证步骤是第一个和XML相关的步骤，我们随后讨论。

验证完后，三个“XML文件输入”步骤并行从XML文件中抽取数据，输入步骤我们以后再

详细讨论, 现在只看数据流。XML文档里的数据最终都要插入到图21-2中的数据库表里。转换要考虑到数据的完整性, 所以要使用正确的顺序加载表, `film_category`和`film_actor`表里的数据要引用到 `category`、`film`和`actor`表里的数据, 所以要先加载被引用表再加载引用表。

另外还有一个比较复杂的技术点就是`sakila`数据库使用了代理键, 代理键是数据库自动生成的。XML文档也使用了代理键来唯一标识`video`和`actor`, 并通过`video`节点内的`actorRef`节点来建立两者之间的关系。可以把XML文档里的代理键认为是业务主键(关于业务主键的概念参考第8章)。

在第8章中, 我们曾讲过, 在数据仓库里, 业务主键可以保存在维度表里, 以便可以通过业务主键查询到代理主键。但是, `sakila`数据库的结构不能从XML中直接加载数据, 没有用来保存XML中业务主键的列。也就是说不能先整体加载`film`、`actor`和`category`表, 再加载`film_actor`表和`film_category`表。如果这样做, 在加载引用表时就不能查找到被引用表的代理键。

在转换里, 通过使用“联合查询/更新”步骤往`actor`、`category`、`film`表里加载数据来解决这个问题。除了把数据加载到相应的表里, “联合查询/更新”步骤还给数据流返回一个数据库生成的代理键。因为`video`节点里还包含了一个 `genre`节点, 所以 `film`表和`category`表使用同一条数据流加载数据(`genre`节点的数据要加载到`category`表中。——译者注), 这就意味着数据加载到`film`表和`category`表后, 数据流中就多了`film_id`字段和`category_id`字段, 这两个字段随后就可以使用一个“表输出”步骤加载到`film_category`表。(转换图中顶部的那条数据流。)

对`film_actor`表来说, 情况稍微复杂一点(见转换中间的数据流)。转换从XML文件里抽取`video`和`actor`的业务主键, `actor`主键来源于 `actorRef`节点。然后使用两个流查询步骤由`video`的业务主键查询出代理主键 `film_id`(上面的数据流), 由`actor`的业务主键查询出代理主键`actor_id`(下面的数据流), 然后使用表输出步骤加载到`film_actor`表里。

使用XSD 验证步骤

如果有XML Schema文件, 最好在处理XML文件之前, 使用XML Schema文件来验证你要处理的XML文件。不做验证可能会导致转换只抽取数据的片段, 甚至把这些不完整的数据加载到数据库, 直到最后才发现XML文档的结构不符合要求。有了XML Schema文件, 就可以提前避免错误的发生。

在Kettle 转换里, 使用“XSD 验证”步骤来验证XML文档是否正确(图12-3里的“Validate against videos.xsd”)。

注意: 和转换里的步骤类似, 也可以在Kettle 作业里使用XSD验证。没有通用的方式说明哪种验证方式更合理。

XSD 验证步骤位于“验证”分类下, 配置窗口如图21-3所示。

配置窗口包含三个部分, 分别介绍如下。

“XML源”

设置要检验的XML文件。XML文件的内容可以全部保存在一个字段里, 这时要设置“XML 字段”下拉列表框。XML文件内容也可保存在一个文件里, 这时字段里保存的不是文件内容, 而是VFS格式的XML文件名, 在这种情况下, 要选中“XML字段值是文件名”选项。



图21-3 使用videos.xsd XML Schema文件验证videos.xml

在图21-3里，没有选中“XML字段值是文件名”选项，说明字段的内容就是一个XML格式的字符串。这个字段的XML内容是通过“Get videos.xml”步骤获得的。

“输出字段”

在“输出字段”部分中，可以设置验证结果字段。在输出流里总是会包括一个验证结果字段，在“结果字段名”输入框里设置保存验证结果的字段。默认情况下，字段名是result，是Boolean类型，使用Y和N分别表示XML是否验证通过。

如果想自己定义验证结果值，不使用Y和N。你需要选中“结果值类型为字符串”选项，并在“XML文件有效时的值”和“XML文件无效时的值”输入框里分别设置相应的返回值。

除了验证结果字段，你还能获得验证的描述信息（通常在验证失败时有用，便于找到验证错误的原因）。如果想保留这些信息，选中“在输出中增加校验消息”选项，并设置保存校验消息的字段名。

“XSD (XML Schema Definition) 设置”

最后是XSD设置，指定用来校验XML文件的XML Schema Definition文件，即XSD文件。“XSD源”下拉列表里指定了XSD文件的位置。

- 来自文件：XSD是一个文件，要直接设置文件名。
- 是一个文件，文件名定义在字段里：XSD是一个文件，但文件名保存在一个字段里，需要选择一个字段。
- 在XML文件里定义：要校验的XML文件本身会有一个xsi:schemaLocation属性，这个属性里包括了XSD文件的URI路径。

在图21-3的例子中，我们使用了第一个选项，并用了一个内置变量\${Internal.Transformation.

Filename.Directory}来指向和转换文件同目录的videos.xsd文件。

注意：尽管也可以动态从Stylus Studio 网站上抽取videos.xsd文件，如同videos.xml文件一样，但我们不推荐这样做。

我们使用xsd文件作为我们转换的XML文件格式标准。如果某个人改变了网站上的xsd文件（同时也改变了videos.xml文件），那么校验XML文件没有问题，但这个XML文件格式已经不是我们期望的格式了。

如果想验证XML文件，你就要确保一旦XML文件格式发生变化，和转换期望的XML格式不一样时，验证步骤就会验证失败。

错误处理

XSD 验证步骤还支持错误处理。如图21-2所示，在验证XML文件时发生的错误都被记录到“validation.error”文件中，使用“XSD检验”步骤下方的“文本文件输出”步骤完成错误输出。错误输出文件只有在校验失败时才有用处——通过错误输出文件查找错误原因，如xsd文件本身格式错误等。

不要把错误处理和XML文件的校验失败混淆起来，XML文件校验失败并不是错误，只是说明XML文件的结构不符合它的结构定义，校验过程实际是成功的。XML文件校验失败只是会把校验结果设置为否，下面讨论这种情况。

检查校验结果

如果你用到了XSD 验证步骤，你肯定需要检查校验结果。XSD 验证步骤会把校验成功和失败的XML文档和相应的校验结果或校验消息都发送到下一个步骤中。ETL设计人员会根据校验结果来决定接下来的处理。

在import_xml_into_db 转换中，使用一个“过滤记录”步骤来验证结果，步骤名是“Is valid? ”，该“过滤记录”步骤验证result字段的值，让result是true的记录通过，过滤result是false的记录，使用“文本文件输出”步骤保存在一个文件里。

使用“XML文件输入”步骤

验证完XML文件之后，就可以开始从XML文件中抽取数据。使用“XML文件输入”步骤来抽取数据，在步骤位于Spoon设计器步骤分类树的“输入”类别下。import_xml_into_db 转换里有三个“XML文件输入”步骤。

- /result/videos/video：从XML文档中抽取视频节点里的数据，每个视频节点的数据将来保存为sakila数据库的film 表里的一行。
- /result/videos/actorRef：每个视频节点里都有不定数量的actorRef节点，来表示在这个电影里有多少个演员。这种电影和演员的对应关系将来保存到film_actor 表里。
- /result/actors/actor：从XML文档中抽取演员节点的数据，将来保存到actor 表中。

这些步骤都抽取同一个XML文档里的不同节点，以并行的方式抽取。出于简化和完整性考虑，这里我们只讨论其中的/result/videos/actorRef步骤。

/result/videos/actorRef步骤的配置对话框里有三个标签页。

“文件”标签

在“文件”标签里（如图21-4所示），可以定义XML文档的数据源。

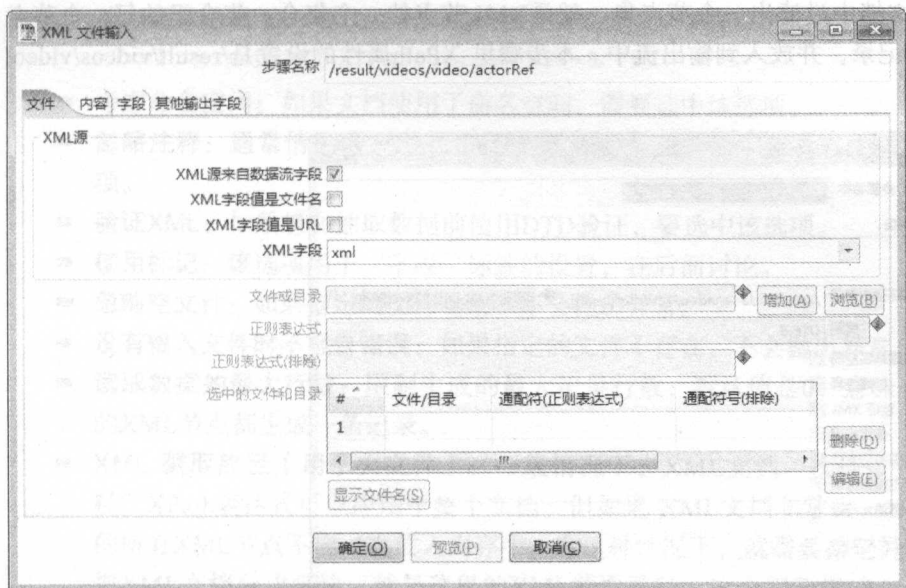


图21-4 “XML文件输入”步骤的“文件”标签

XML文档的数据源可以通过输入数据流里的字段指定，也可以通过标准的目录文件的方式指定。如果要使用字段的方式，选中“文件”标签页上方的“XML源来自数据流字段”复选框。如果不选中这个复选框，就要在对话框的下半部分直接设置文件名或设置目录和正则表达式来过滤文件。

注意：设计转换时最好不要选中“XML源来自数据流字段”选项，而应使用目录文件的方式。使用文件的方式可以简化流程设置，使用文件方式 Kettle可以定位到这个文件，解析这个文件，在做XPath 设置时，还可以进行提示。如果使用字段方式，字段内容只有在运行时才知道，在设计时是不知道的。

可以任意切换文件和字段这两种数据源方式。

如果选中了“XML源来自数据流字段”，就要在“XML字段”下拉列表框里选择一个字段。最后还要选择Kettle 如何解释XML字段里的内容，有以下三种解释方式。

- 如果选中了“XML字段值是文件名”选项：说明选中的XML字段里的值是一个文件名。
- 如果选中了“XML字段值是URL” 选项：说明选中的XML字段里的值是一个URL 而不是文件名。
- 如果上面两个选项都没有选中，说明字段内容就是XML格式的文本。

在import_xml_into_db 转换里，我们用了上面的最后一个方式，在前面的步骤里先用HTTP 请求把整个XML内容抽取过来，然后再用三个并行的“XML文件输入”步骤同时抽取文档里的片段。如果用其他方式，则还要再做HTTP 请求，比较慢。

“内容”标签

在“内容”标签里可以指定如何从XML文档里抽取数据行。import_xml_into_db转换的/result/

videos/video/actorRef步骤的“内容”标签如图21-5所示。

标签里最重要的属性就是“循环读取路径”。在这里需要设置一个XPath 表达式。XPath 表达式将从 XML文档中过滤出一个节点集，就是XML节点的一个集合。集合里的每一个节点都将被解析为一行记录，并放入到输出流中。本步骤里 XPath属性的设置是/result/videos/video/actorRef[text()]。

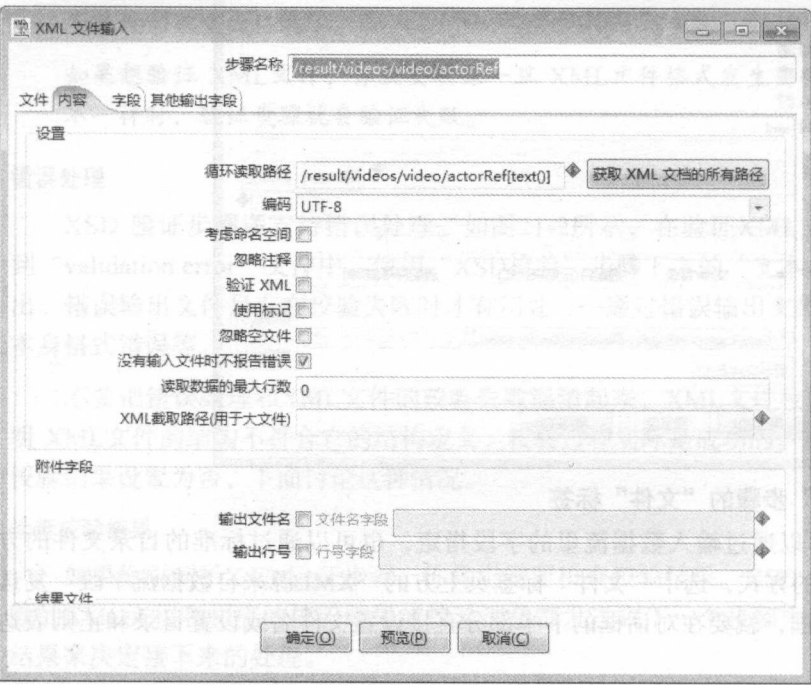


图21-5 “XML文件输入”步骤的“内容”标签

如果你已经在“文件”标签里指定了一个XML文件，你就可以使用“获取XML文档的所有路径”按钮来帮助你设置 XPath属性。这个按钮获取了XML文档里的全部路径（如图21-6所示）。

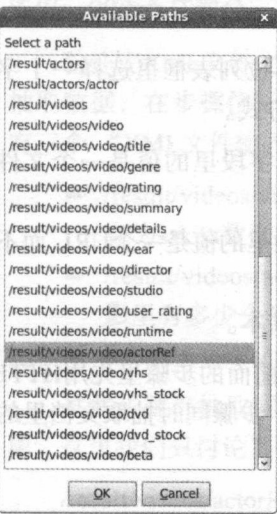


图21-6 获取全部路径的选择列表

“内容”标签里的其他属性还包括以下一些。

- **编码**：用来设置XML文档的编码。如果XML文档本身没有指定编码，就要用到这个选项。通常情况下，XML文档的编码在文件头定义，例如：
`<?xml version="1.0" encoding="UTF-8"?>`
- **考虑命名空间**：如果文档使用了命名空间，需要选中该选项。
- **忽略注释**：通常情况下，XML注释也看作是节点，如果要忽略注释节点，要选中该选项。
- **验证XML**：如果想在抽取数据前使用DTD验证，要选中该选项。
- **使用标记**：该选项用于“字段”标签的设置，在后面讨论。
- **忽略空文件**：如果指定的文件是空，不会抛出异常。
- **没有输入文件时不报告错误**：如果指定的文件不存在，不会抛出异常。
- **读取数据的最大行数**：限制生成的最大记录行数，默认值是0，意味着对每一个抽取到的XML节点都生成一条记录。
- **XML 截取路径（用于大文件）**：一般情况下，XML文档一次性读入内存，“读取路径”XPath表达式可以应用于整个文档。但如果XML文档非常大，XPath表达式匹配到的所有XML节点不能一次放入内存中，在这种情况下，就需要指定另一个XPath表达式把XML文档分成多块，就是这里的XML截取路径。这个用于把XML文档分块的XPath路径不支持全部的XPath语法，只能使用斜线分隔的节点名这种语法格式，不支持命名空间和谓词表达式。另外截取路径XPath必须是读取路径XPath的上一级或同级目录。
- **输出文件名/文件名字段**：如果使用XML文件作为源，“输出文件名”选项可以在输出流中增加一个字段保存XML文件名。“文件名字段”选项设置新增字段的字段名。
- **输出行号/行号字段**：“输出行号”选项可以为每一个数据行生成的一个序列号，“行号字段”选项设置行号字段的字段名。
- **将文件添加到结果文件列表**：如果使用了XML文件，选中该选项把文件添加到结果文件列表中。你在父作业中就可以再处理这个文件。

“字段”标签

在“内容”标签中已经使用XPath表达式匹配了XML节点集。“字段”标签（如图21-7所示）用来从XML节点中抽取字段。

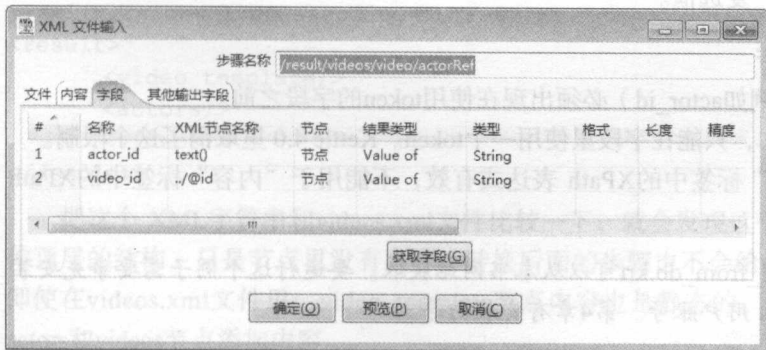


图21-7 “XML文件输入”步骤的“字段”标签

如图21-7所示，在该标签中也使用XPath表达式来抽取字段。图中表格中的“名称”列用来设置要抽取的字段名。在“XML节点名称”列里，使用XPath表达式指定从哪里获得字段的

值。XPath 表达式用来匹配 XML数据行里的字段。

通常情况下，图21-7表格中的“节点”列设置为“节点”。如果只想匹配 XML文档里的属性，“节点”列需要设置为“属性”。例如图中的第二行使用了“../@id” XPath 表达式，该表达式代表当前节点父节点的id属性。如果把“节点”列设置为“属性”，同时把XPath 设置为../id，也能达到同样的效果。如果把 ../id 当成了普通的XPath 表达式，就代表了当前节点父节点的id 子节点的值，其含义是完全不一样的。

在图21-7中，actor_id字段从actorRef节点中抽取文本内容。video_id字段使用 ../@id 表达式，从actorRef节点的父节点video中，抽取其 id属性。

现在输出流中就包括了video和actor两个标识字段。在转换的后面会用到这两个标识字段去查询数据库表里的键值，并最终用来向 film_actor 表里添加新行。

使用token

“字段”标签里的XPath 表达式支持一种非标准化的称为token的扩展形式。token用来参数化XPath表达式，它可以把字段的值绑定到 XPath表达式里。我们下面来看一个例子。

我们继续看 import_xml_into_db 转换里的/result/videos/video/actorRef步骤，假设除了actor_id 和video_id字段，我们还想抽取演员的名字。如果抽取了actorRef节点的文本内容，我们实际就获得了actor节点的id属性，这样看上去我们就应该可以抽取到对应的actor节点里的演员名字。

例如，抽取到的actor_id是1234，那么/result/actors/actor[@id=1234]/text() 就会抽取到演员的名字。但现在的问题是，1234是在运行时才知道的。在设计时只知道actor_id，在当前的actorRef 节点下使用 text() 表达式可以获得actor_id的值。

但我们不能直接把 1234 替换成text()，/result/actors/actor[@id= text()]/text()这样的XPath 不能工作。这样的表达式的含义是匹配节点的id属性等于节点内容的演员节点。这里的text()是actor 节点的内容，而不是actorRef节点的内容。因此我们需要一种机制，可以把actor_id的值放入到 XPath表达式里，这也就是token的用途。

为了要在 XPath 表达式里使用 actor_id字段，要把表达式写成/result/actors/actor[@id=@_actor_id]/text()。表达式@_actor_id就是token，这个token 包括一个at 符号(@)，一条下划线(_)，然后是字段名actor_id，最后是一个短横线(-)。另外，如果要使用token，需要选中“内容”标签里的“使用标记”复选框。

使用token 有下面几个限制：

- token 里使用的字段（例如actor_id）必须出现在使用token的字段之前。
- 在Kettle 4.0 以前的版本，只能在字段里使用一个token。Kettle 4.0 里取消了这个限制。
- token 语法只对“字段”标签中的XPath 表达式有效，不能用于“内容”标签中的XPath 表达式。

注意：转换文件export_xml_from_db.krt可以从本书网站获取。要运行这个例子需要事先安装好sakila例子数据库和sakila 用户账号。第4章有安装向导。

21.3.3 生成XML文档

为了要把数据从 sakila数据库导出到videos.xml这种格式的文件里，我们设计了export_xml_

from_db 转换, 如图21-8所示。

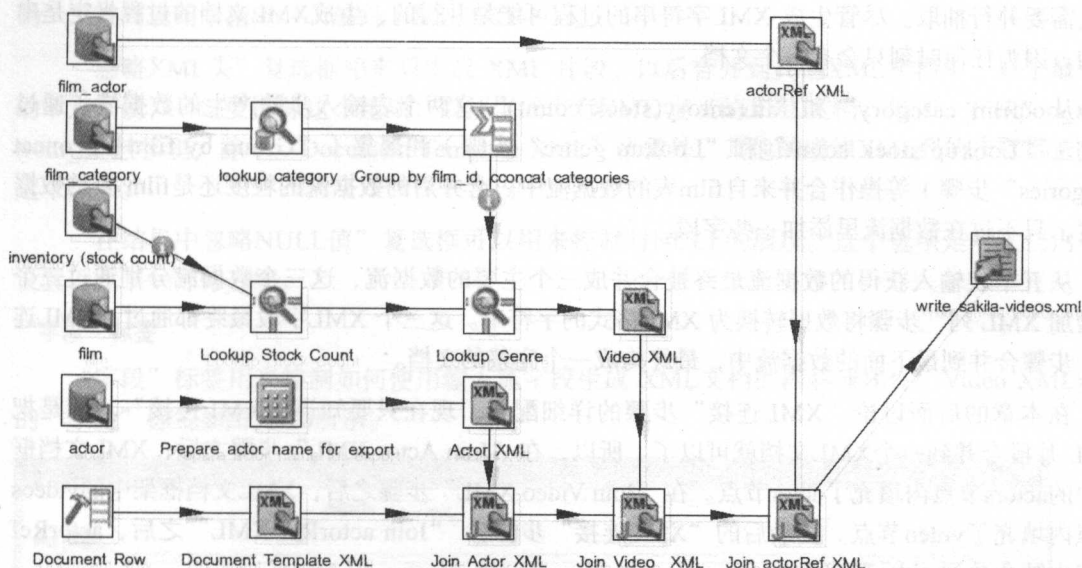


图21-8 生成XML文档

Kettle 里有两个步骤专门用来生成XML格式文档, 这两个步骤分别是“转换”类别下的“增加 XML 列”步骤和“连接”类别下的“XML连接”步骤, 上面的转换里用了很多次这两个步骤。在看 export_xml_from_db 转换的每个步骤之前, 我们先看看这个转换的主要数据流。

export_xml_from_db 转换的总体设计

图21-8 转换里的主要的数据流是最下面的那条数据流, 它是构造XML文档的最主要的数据流。

该条数据流从一个生成记录步骤开始, 该步骤生成的一个数据行包含三个空字符串字段 actors、videos和video_template。该行随后被发到了下面“Document template XML”步骤, 这是一个“增加 XML 列”步骤, 在后面讨论这个步骤的详细配置, 现在只看这个步骤的输出结果, 这个步骤将在输出流中生成一个新的字段, 包含一个内容为空但结构完整的XML格式字符串。

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <video_template/>
  <actors/>
  <videos/>
</result>
```

把这个 XML字符串和videos.xml文件比较一下, 就会发现这个XML字符串就是videos.xml文件顶层的结构, 只是节点里没有内容。转换后面的步骤也不会给video_template节点添加内容。即使在videos.xml文件里, video_template节点内容也是静态的。所以下面我们主要说明如何给actors和videos节点添加内容。

转换里除了最下面的数据流外, 其他数据流都以“表输入”步骤作为起始步骤, 这些数据流以并行方式从sakila数据库中抽取数据。可以发现, 图21-1 里的所有表都在这个转换里, 除此之外, 还有一个额外的步骤来抽取每个电影的库存数。

一般在真实的转换里，不需要太多的并行抽取。在这里我们只要注意，构造XML文档并不一定需要并行抽取。尽管生成 XML字符串的过程可能是并行的，生成XML文档的过程肯定是串行的，因为任何时刻只会有一个文档。

从“film_category”和“inventory(stock count)”这两个表输入步骤产生的数据流，通过查询（“Lookup stock count”和“Lookup genre”步骤）和聚集（“Group by film_id,concat categories”步骤）等操作合并来自film表的数据流中。合并后的数据流的粒度还是film 表的数据粒度，只不过在数据流里添加一些字段。

从五个表输入获得的数据流最终被合并成三个主要的数据流，这三个数据流分别通过三个“增加 XML 列”步骤将数据转换为 XML格式的字符串。这三个 XML字段最终都通过“XML连接”步骤合并到最下面的数据流中，最终构成一个完整的文档。

在本章的后面讨论“XML连接”步骤的详细配置，现在只要知道“XML连接”步骤是把XML 片段合并到一个XML文档就可以了。所以，在“Join Actor XML”步骤之后，XML文档框架中的actors节点内填充了actor节点。在“Join Video XML”步骤之后，XML文档框架中的videos节点内填充了video节点。在最后的“XML连接”步骤，“Join actorRef XML”之后，actorRef节点也被合并到 video节点中。

转换的最后一个步骤使用一个普通的文本文件输出步骤，将XML文档输出到一个文件中，最后形成一个包含sakila库中所有电影相关数据的XML文件。

使用“增加 XML 列”步骤生成XML

“增加 XML 列”步骤用于生成XML节点。该步骤在设计器左侧面板树下面的“转换”类别下。对输入流里的每一行，“增加 XML 列”步骤会添加一个包含XML字符串的新字段，并把这一行发送到下一个步骤中。在配置对话框里，可以设置生成的XML节点的名字、属性、内容等。配置对话框有两个标签：内容和字段。

“内容” 标签

图21-9是export_xml_from_db 转换中“Video XML”步骤的“内容”标签。

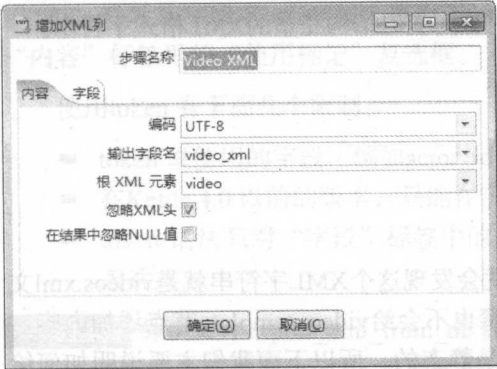


图21-9 Video XML步骤的“内容”标签

“内容”标签这个名字有一点令人迷惑，这个标签实际用于设置生成的XML节点的属性，而不是它的内容。

“输出字段名”属性设置保存XML节点的字段名。“根XML元素”属性设置XML节点的

名称。注意，节点名称目前是一个字符串常量，不能指定一个字段来动态设置节点名称。“编码”下拉列表用来指定一个编码（默认 UTF-8）。

“忽略XML头”复选框用来只生成 XML 片段，以后合并到其他XML文档中。对于最外层的节点来说，一定要清除这个选项，以便生成带有XML 定义的XML文档。例如，在export_xml_from_db 转换里，除了“Document Template XML”步骤外，其他的增加XML列的步骤都选中了这个选项。

“在结果中忽略NULL值”复选框可以用来控制对NULL的展现。这个选项是对文档内容的设置，在下一部分介绍。

“字段”标签

“字段”标签用来控制如何使用输入流字段生成 XML文档的内容或属性。Video XML步骤的“字段”标签如图21-10所示。



图21-10 Video XML步骤的“字段”标签

表格里的每一行都代表了输入流里的一个字段，这些字段用来构造XML文档。“节点名”属性用来设置你要使用哪个字段。输入流字段可以通过四种方式来构成 XML文档。

- 生成“Root XML element”的子节点，把字段内容作为子节点内容。表格中的“节点名”用来设置生成的节点名。例如图21-10中的“description”字段使用了两次，分别用来生成summary节点和details节点。如果不设置“节点名”，字段名将作为节点名。例如，title、genre和rating字段会生成同名的XML节点。
 - 生成“Root XML element”的属性，把字段内容作为属性的内容。这种设置方式需要把表格里的“属性”列设置为Y，并把“属性的父名称”列留空。在这个例子里，“节点名”将作为属性名。如果没有设置“节点名”，“字段名”将作为属性名。例如，图21-10中的film_id字段就设置为“Root XML element”的属性。
 - 把字段内容作为“Root XML element”的文本内容。这种方式的配置和上面的第一种方式的配置非常类似。唯一的不同之处是必须使用“Root XML element”的名字作为“节点名”的名字。尽管配置变化不大，最后效果相差却很大：不会生成子节点，字段的值作为“Root XML element”节点的内容。“actorRef”步骤是这种设置方式的例子。
- “actorRef”步骤的“字段”标签只有一个“actor_id”字段，该字段的“节点名”被设

置为actorRef，在“内容”标签中，“Root XML element”也被设置为actorRef。最后生成的结果就是<actorRef>1234</ actorRef>，1234是actor_id字段的值。

- 生成“Root XML element”的子节点，把字段内容作为子节点属性。这种方式的配置和第二种方式类似。不同之处就是需要在“属性的父名称”中输入要设置属性的节点的 名字。

如果字段有NULL 值，默认情况下会产生一个空节点或属性值。可以选中“内容”标签中“在结果中忽略NULL值”选项来忽略这样的节点或属性值。

下面是一个输入数据行，通过“Add Video XML”步骤后生成的XML文本的例子：

```
<video id="2">
  <title>ACE GOLDFINGER</title>
  <genre>Horror</genre>
  <rating>G</rating>
  <summary>
    An Astounding Epistle of a Database Administrator
    And a Explorer who must Find a Car in Ancient China
  </summary>
  <details>
    An Astounding Epistle of a Database Administrator
    And a Explorer who must Find a Car in Ancient China
  </details>
  <year>2006</year>
  <dvd>12.99</dvd>
  <dvd_stock>3</dvd_stock>
  <actorRef>19</actorRef>
  <actorRef>85</actorRef>
  <actorRef>90</actorRef>
  <actorRef>160</actorRef>
</video>
```

使用“XML 连接”步骤

尽管“增加 XML 列”步骤可以生成XML文本，但它也只能生成一层 XML文本，另外生成的XML的子节点是固定的，这些子节点都需要事先知道。“XML 连接”步骤可以解决这些问题。

“XML 连接”步骤用来把多个 XML文本合并到一个XML文本中，最终形成一个单一的XML文本（目标）。尽管这个步骤是把两个XML文本连接起来，但不要把XML 连接和数据库连接相比较。我们看下面的例子。

我们看一下 export_xml_from_db 转换里的“Join Actor XML”步骤，这个步骤的配置界面如图21-11所示。

目的流属性

在本例中，合并的目的流是空模板文档的数据流，空模板由转换图下方的“Document Template XML”步骤生成，并发送到“Join Actor XML”步骤中。“目的XML步骤”设置目标数据流，“目的XML字段”设置目标数据流里的XML字段。

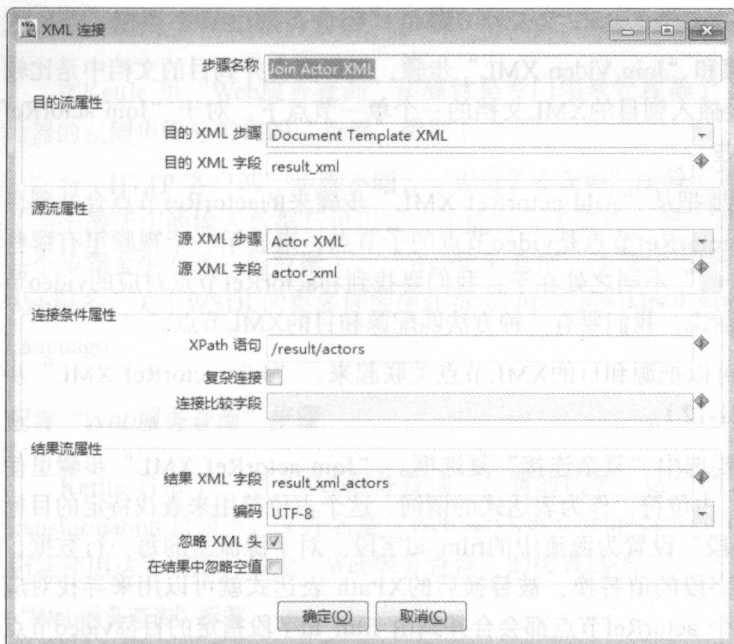


图21-11 Join Actor XML步骤的配置界面

源流属性

源流是要被合并到目的流中的数据。这里通过“源 XML 步骤”将“Actor XML”步骤设置为源流，源流里定义了一组演员节点（一行是一个演员）。“源 XML 字段”设置源流里的哪个字段是要合并到目标的 XML 字段。

连接条件属性

连接条件属性用来控制源流的 XML 文档要加入到目的流 XML 文档的哪个节点之下。这里需要指定一个 XPath。在图 21-11 中，“XPath 语句”是 /result/actors，就是说从源流来的每一行 actor 节点都将插入到目的 XML 的 /result/actors 节点下。

结果流属性

在结果流属性中，可以设置保存合并结果的字段名称。另外还可以设置编码，是否忽略 XML 头、如何处理 NULL 值等。这些属性和“增加 XML 列”步骤里的同名属性类似。

下面是经过“Join Actor XML”步骤之后的 XML 文档的内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <video_template/>
  <actors>
    <actor id="1">Guinness, Penelope</actor>
    ...many more actor elements...
    <actor id="884">Sharif, Omar</actor>
  </actors>
  <videos/>
</result>
```

复杂连接条件

对于“Join Actor XML”步骤和“Join Video XML”步骤，把源流合并到目的文档中是比较简单的：从源流来的XML 都直接插入到目的XML文档的一个单一节点下。对于“Join actorRef XML”步骤来说，情况要复杂一些。

“Join actorRef XML”步骤要把从“Add actorRef XML”步骤来的actorRef节点合并到目的文档中。我们以前曾经说过，actorRef节点是video节点的子节点，来表示一个视频里有哪些演员。所以和其他“XML 连接步骤”不同之处在于：我们要找到和actorRef节点对应的video节点，再插入到video节点下。换句话说，我们要有一种方法匹配源和目的XML节点。

这里所谓的复杂连接条件就可以把源和目的XML节点关联起来。“Join actorRef XML”步骤里就使用了这样的条件（见图21-12）。

要使用复杂连接条件，首先要选中“复杂连接”复选框。“Join actorRef XML”步骤里使用的XPath 表达式里有一个“?” 占位符，作为表达式的谓词，这个占位符用来查找特定的目标video节点。另外，“连接比较字段”设置为源流中的film_id字段。对于源流里的每一行数据，XPath 里的占位符都会被film_id字段的值替换，被替换后的XPath 表达式就可以用来寻找对应的video节点。通过这种方式，每个 actorRef节点都会合并到由 film_id字段指定的目标video节点中。和上面讲的“XML文件输入”步骤一样，这里的“?” 占位符，也是非标准的XPath。

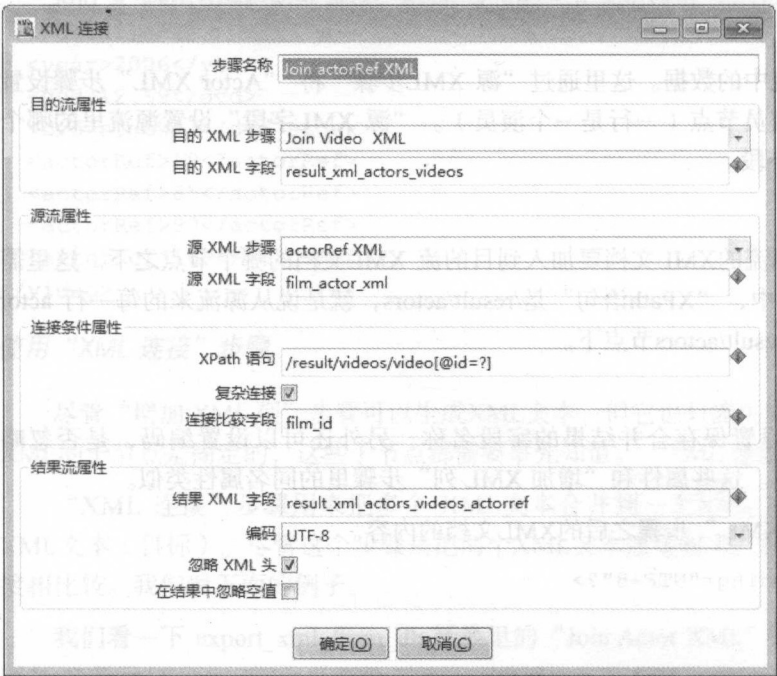


图21-12 “Join actorRef XML” 步骤中，使用复杂连接把源和目的XML 关联起来

21.4 SOAP例子

在本节，我们讲述如何使用“Web服务查询”步骤来调用基于SOAP的Web服务。

21.4.1 使用“Web服务查询”步骤

在Kettle里“Web服务查询”步骤就是专门用来处理基于SOAP的Web服务的。该步骤在设计器的左侧步骤分类树的“查询”类别下。

与“HTTP客户端”步骤不同，“Web服务查询”步骤不需要前面有一个输入数据行来激活（一个激活用的输入数据行可以是“生成行”步骤产生的，也可以是其他任何步骤，只要往下一个步骤至少发送一行数据）。“Web服务查询”步骤使用Web服务描述语言（WSDL）来获取Web服务。关于WSDL的更多详细描述参考http://en.wikipedia.org/wiki/Web_Services_Description_Language。

配置“Web服务查询”步骤

Kettle自带了一个“Web服务查询”步骤的例子。这个例子在Kettle安装目录下的 samples/transformation 目录下，文件名是“Web services lookup-convert degrees Celsius to Fahrenheit”。我们就使用这个例子，讨论“Web服务查询”的配置选项。

“Web服务查询”标签

“Web服务查询”步骤配置对话框打开后，都会有一个“Web服务查询”标签（见图21-13）。

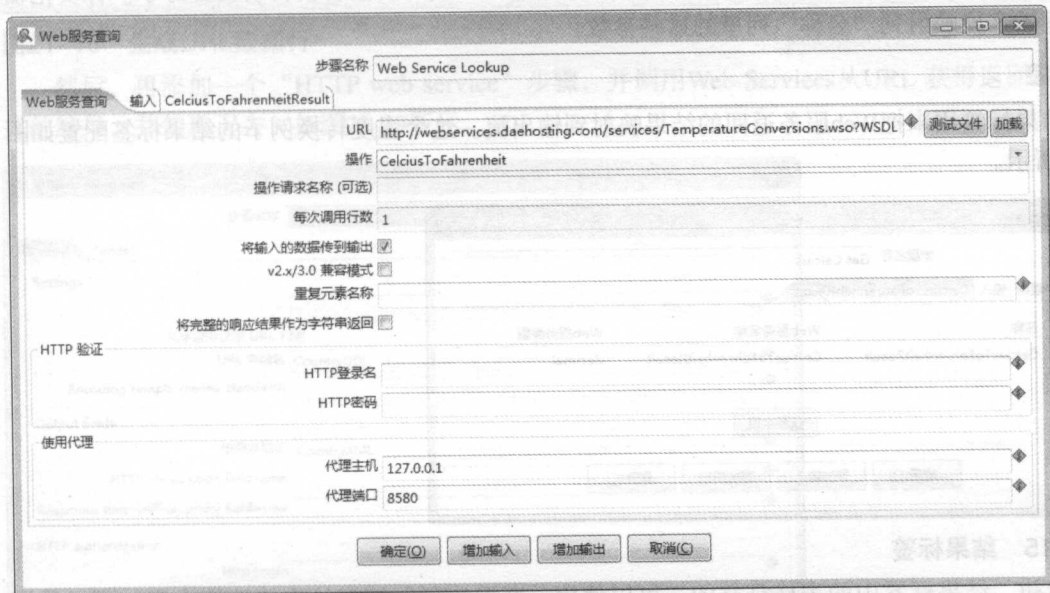


图21-13 “Web服务查询”步骤配置对话框

这个标签下最重要的一项就是URL，用来输入一个可以返回WSDL的URL。如果Web服务还需要授权，需要在对话框中间的“HTTP验证”栏内输入“HTTP登录名”和“HTTP密码”。如果你需要使用代理访问Internet，还要在对话框下面的“使用代理”栏中，设置“代理主机”和“代理端口”。

输入了URL以后，单击“加载”按钮，该步骤发出一个获取WSDL的请求，并自动给“操作”下拉列表填充可选项。在本例中，自动填充的操作可选项有四个：CelciusToFahrenheit、FahrenheitToCelcius、WindChillInCelcius和WindChillInFahrenheit。这些是这个Web服务提供的方

法,你要从中选择一个操作。

选择一个操作后,对话框会自动创建两个标签,一个用来指定输入参数,另一个用来控制如何处理Web服务的返回结果。

“输入” 标签

“输入” 标签是选中一个需要参数的操作后,自动产生的标签。标签里有一个表格用来填写对选中的操作的输入参数。这个温度转换例子的“输入” 标签如图21-14所示。

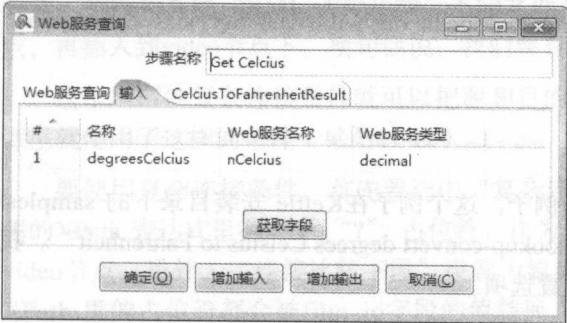


图21-14 “输入” 标签

“Web服务名称” 和 “Web服务类型” 列在单击“加载” 按钮后就自动设置好了。你要把输入流里的字段映射到“名称” 列里的这些参数。

结果标签

结果标签用来把Web服务返回的结果映射到输出流。这个温度转换例子的结果标签配置如图21-15所示。

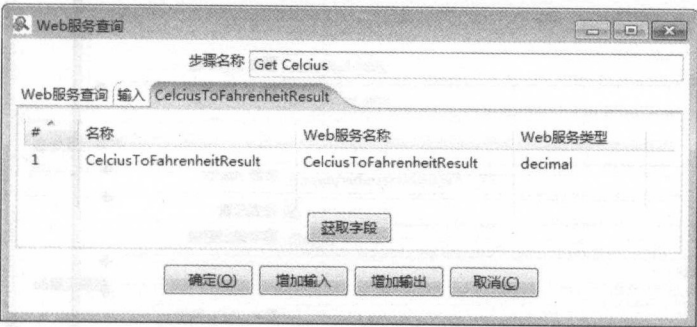


图21-15 结果标签

最初,结果标签中的表格是空的,可以使用“获取字段” 按钮给表格填充内容。可以使用“名称” 列把Web服务的返回结果映射成输出流的字段。

21.4.2 直接访问 SOAP服务

WSDL可以支持不同的实现和方言,目前Kettle 只支持其中的一部分实现。如果“Web服务查询” 步骤没有按预期工作,如返回空结果或返回错误信息,可以选择“将完整的响应结果作为字符串返回” 复选框来接收原始的XML格式报文,并使用“XML文件输入” 步骤来解析。另外,还可以直接使用“HTTP client” 和“HTTP Post” 步骤来访问服务并使用“XML文件输入” 步骤来解析。

一般通过 Web Services来获取一些基础数据，如国家编码和国家名称。也有很多公开的Web Services，用来提供基础数据。例如用来提供国家信息的Web Services，<http://www.oorsprong.org/websamples.countryinfo/CountryInfoServices.wso>。这个Web Services 列出了很多操作，如 ListOfCountryNamesByName，将在下面的例子里使用。

提示：关于测试和使用 Web Services，除了Kettle，我们还推荐一款工具，就是开源的 soapUI。关于该工具可以参考 <http://www.soapui.org>。

在Kettle 里使用 Web Services有一个很大的问题：为了正确解析XML，需要事先知道Web Services 的输出格式。这就是soapUI这种工具的作用。也可以使用Kettle 里的“预览”选项来查看输出。创建一个新转换并添加一个“生成记录”步骤，该步骤产生一个带 URL字段的数据行，如图21-16所示。



图21-16 生成URL数据行

然后，再添加一个“HTTP web service”步骤，并调用Web Services从URL 获得返回XML格式的结果，并传递下去，如图21-17所示。

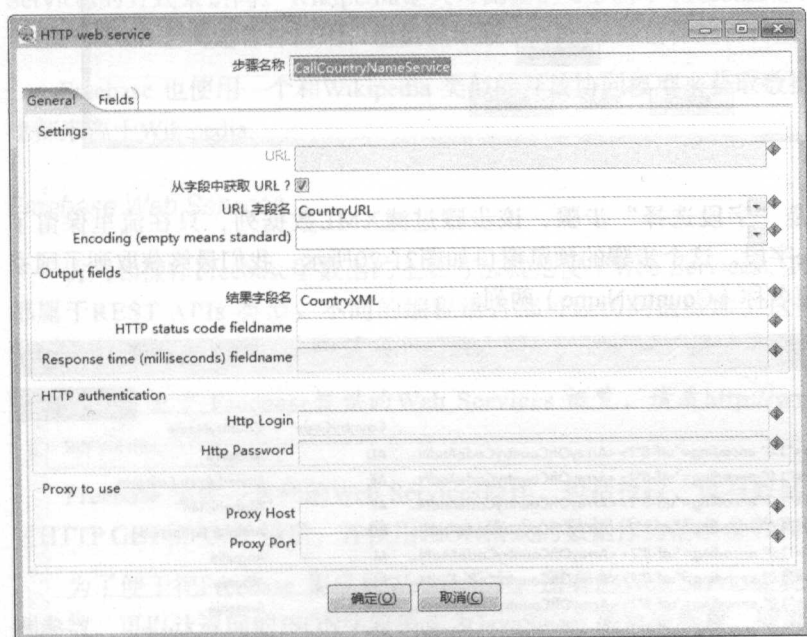


图21-17 调用 Web Services

说明：“HTTP web service”发出了一个HTTP GET请求，一些SOAP Web Services可能不会响应 GET方式的请求，只能响应 POST方式的请求。如果这样，可以试试“HTTP Post”步骤。“HTTP Post”步骤也在设计器左侧面板的“查询”类别下。

为了让后面的“XML文件输入”步骤的输入字符串格式正确，需要知道实际的运行结果是什么。可以使用“HTTP web service”步骤的预览功能来查看数据格式。如图21-18所示，结果是一个 XML数据行，根节点是ArrayOfCountryCodeAndName，下面包含多个重复的tCountryCodeAndName节点。每个tCountryCodeAndName节点又包含sISOCode和sName节点。

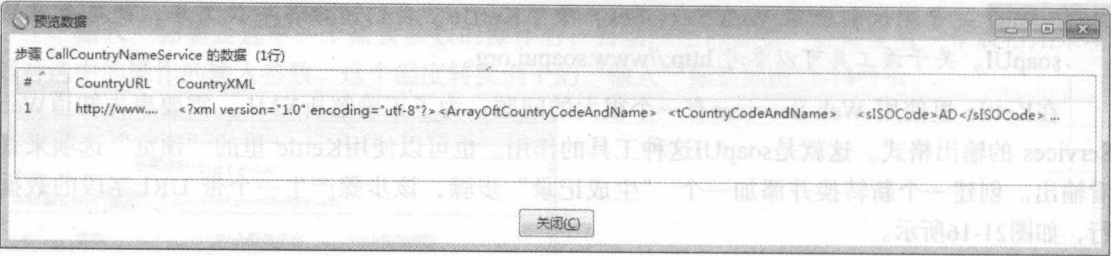


图21-18 查看预览结果

得到预览结果后，就可以解析了。首先在“XML文件输入”步骤中选中第一个复选框（“XML定义在字段里”）并指定CountryXML 作为源。然后，在“内容”标签下，将查询XPath 设置为 ArrayOfCountryCodeAndName/tCountryCodeAndName。最后在“字段”标签里设置要抽取节点的XPath和对应的字段名，如图21-19所示。

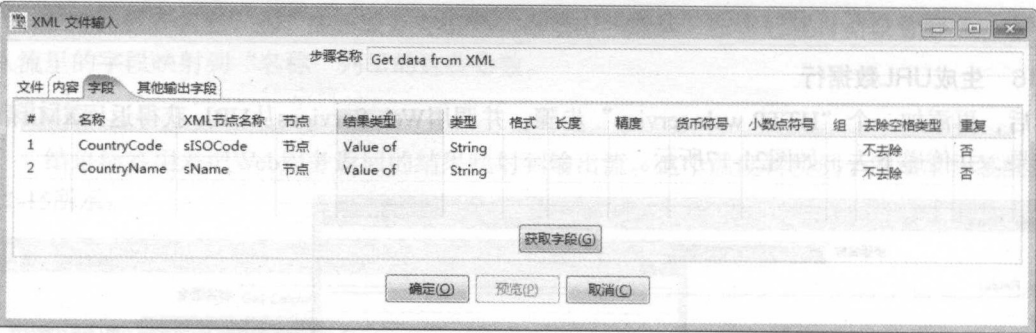


图21-19 XML文件输入

例子里最后一个步骤是“字段选择”步骤，该步骤过滤XML数据列，只在流里保留了CountryCode和CountryName字段。这个步骤的预览窗口如图21-20所示，我们最终获取到了国家代码（CountryCode）和国家名称（CountryName）两列。

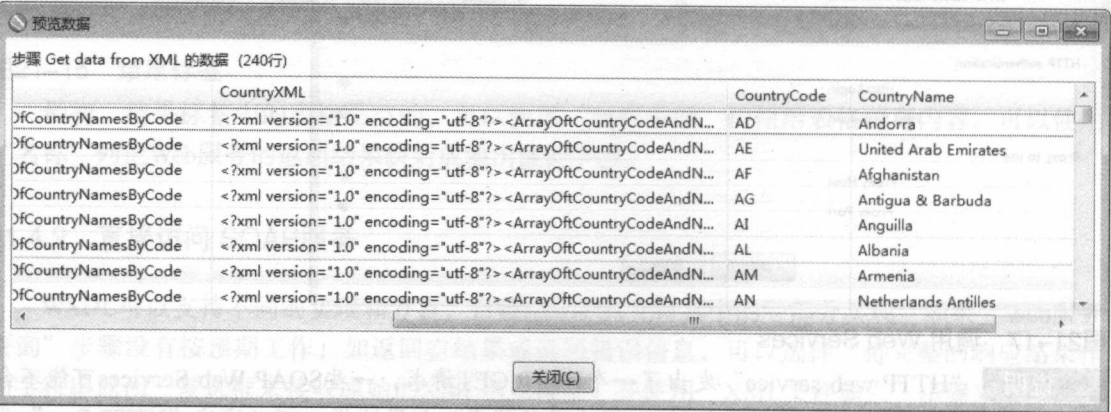


图21-20 国家代码和国家名称预览窗口

21.5 JSON例子

我们前面已经说过了，只能通过Java脚本步骤来解析JSON字符串。我们还没有讨论如何在JavaScript Value步骤里访问它们，并把它们转换为数据行。

我们下面详细说明一个例子，从Freebase数据库项目中抽取JSON数据。我们先介绍Freebase项目的一些背景知识，它的Web Services和这些Web Services如何使用JSON来工作。

21.5.1 Freebase项目

将根据Freebase项目提供的数据和服务来介绍我们的例子。Freebase项目是一个大型数据库，包括了各种主题信息。

Freebase项目由Metaweb公司资助，该公司拥有Freebase使用的数据库软件。而Freebase数据在Creative Commons协议授权下，任何人都可以访问。

说明：关于Freebase可参考<http://wiki.freebase.com/wiki/FAQ>。关于Metaweb，参考<http://www.metaweb.com/faq>。

Freebase和Wikipedia

Freebase类似于Wikipedia，它也是一个开放的协同操作的百科全书。和Wikipedia不同的是，它的数据是严格按照关系和实体来构造的，而且这些实体和关系可以使用查询语言以Web Services的方式来访问。Wikipedia是人可阅读的知识库，Freebase是知识库的框架，开发人员可以在Freebase上构建各种应用，包括但不限于Wikipedia这样的百科应用。

Freebase也使用一个和Wikipedia类似的开放协同模型来获取数据。在Freebase里的很多主题也都来源于Wikipedia。

Freebase Web Services

访问和操作Freebase里数据的主要方法就是使用Web Services。Freebase提供的Web Services都属于REST APIs类型。不同的编程语言都可以访问这些库，这些库提供的API封装了Web Services，隐藏了发送HTTP请求、接收响应、处理返回结果的细节过程。

说明：关于Freebase提供的Web Services概览，请看http://www.freebase.com/docs/web_services。

Freebase提供一系列的Web Services操作，包括授权、读写数据、搜索等。所有的操作都通过HTTP GET和POST请求，并使用JSON格式的数据作为请求参数和响应结果。

为了便于把Freebase集成到Web应用中，所有的Web Services都支持一种回调参数。指定回调参数，可以让返回的JSON字符串作为JavaScript函数的参数，回调参数指定了函数名。这种技术也被称为JSONP，在需要自动处理服务器响应的Web应用里广泛使用。当使用JavaScript来处理服务器响应时，回调函数被调用，并使用服务器返回的JSON作为参数。

Freebase的读服务

我们的例子集中在 Freebase 的读服务, 就是从 Freebase 中获取数据。读服务可以通过 HTTP GET 或 POST 方式访问 URL: <http://api.freebase.com/api/service/mqlread>。直接在浏览器里输入该 URL 即可, 得到下面的响应:

```
{
  "code": "/api/status/error",
  "messages": [
    {
      "code": "/api/status/error/input/invalid",
      "info": {
        "value": null
      },
      "message": "one of query=, or queries= must be provided"
    }
  ],
  "status": "400 Bad Request",
  "transaction_id":
    "cache;cache01.p01.sjc1:8101;2010-04-06T17:36:16Z;0050"
}
```

服务器返回的响应是人可读的 JSON 格式, 这里返回的结果是一个错误信息, 说明请求的格式不正确。这是因为读服务需要指定一个查询, 指定要返回什么数据。

The Metaweb Query Language

Freebase 查询必须使用 Metaweb Query Language (MQL) 语言来查询, 并通过一个 query 参数来传递。所以 URL 应该像下面这样:

<http://api.freebases.com/api/service/mqlread?query=...mql query here...>

MQL 查询本身也是 JSON, 这种查询方法通过提供数据的一个样例来查询数据。关于 MQL 的详细说明不在本书范围, 我们只演示一个简单查询。下面的代码片段是一个有效的 MQL 查询, 查询在 Freebase 里所有的电影导演:

```
[{
  "type": "/film/director",
  "name": null
}]
```

正如你看到的, 查询本身也是一个 JSON 对象, 包含了两个成员, type 和 name。这个对象就像你想要查找的对象的一个例子或模板。

要执行这个查询, 需要把这个查询封装到另一个 JSON 结构里, 我们把这个封装后的 JSON 称为“查询包”, 把这个“查询包”拼接到 Freebase 查询 URL 的后面, 最后像下面这样:

```
http://api.freebase.com/api/service/mqlread?query=
{"query":[{"type": "/film/director", "name": null}]}
```

上面的 URL 把 MQL 查询另起了一行, 实际应该是一行。在浏览器里执行后, 得到下面的结果:

```
{
  "code": "/api/status/ok",
```



```

"result": [
  {
    "name": "Blake Edwards",
    "type": "/film/director"
  },
  ...many more directors go here...
  {
    "name": "Andrew Stanton",
    "type": "/film/director"
  }
],
"status": "200 OK",
"transaction_id":
  "cache;cache03.p01.sjc1:8101;2010-04-06T18:51:11Z;0023"
}

```

说明：出于性能和扩展性考虑，默认情况下，Freebase 限制最多100 个返回结果。通过发送多个请求可以获得更多的结果。关于如何获取更多的结果可以参考 MQL 引用向导 <http://www.freebase.com/docs/mql/ch04.html#cursors>。

我们看到，响应也是一个JSON对象，我们通常也称之为“结果包”。“结果包”里包含了一些属性如code、status和transaction_id，用来承载查询的结果信息。当你第一次使用服务时，你就会接触到这些属性。

MQL查询会被封装到“查询包”里，查询语句作为“查询包”里的query成员。MQL查询结果会被封装到“结果包”里，查询结果作为“结果包”里的result成员。如果把查询和结果对比一下，会发现查询和结果的JSON 结构是一种映射关系：查询和结果都是一个JavaScript数组，查询结果数组里的每个元素的属性和查询语句的属性一样。这种对比如图21-21所示。

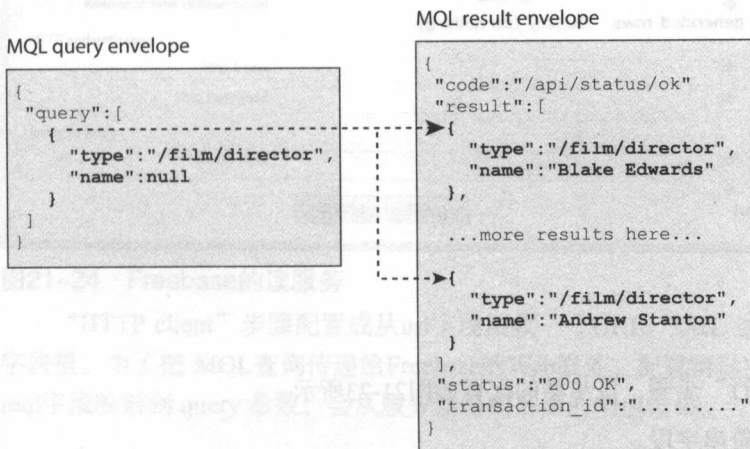


图21-21 MQL查询和结果包

在查询里，type属性说明要查找哪一个实体，如同SQL 语句里的FROM 子句。这个属性将在result里重复。查询给name属性指定了一个null 值，在MQL 里，这意味着要在查询结果里给这个属性赋值，类似于SQL查询中的SELECT 子句。

说明：尽管在某种程度上，MQL和SQL 有相似之处，但我们要注意Freebase 不是关系型数据库。相似之处只是表面上的，只是为了便于你快速理解MQL。

既然我们提到了MQL和SQL的一些相似之处，你可能想知道MQL 里的哪部分和SQL 里的WHERE 子句相似。MQL 没有提供像SQL 那样的条件过滤，但是它支持查询例子，来找出具有给定属性值的数据，只要在查询里指定这个属性的值。执行查询后返回的结果，将自动过滤出那些和查询例子里属性值一样的对象，并把其他属性为 null 的值替换成查询出的值。

我们刚才讨论的MQL 基础知识，对下面的Kettle例子应该足够了。

说明：MQL查询语言比我们刚才讨论的要复杂得多：MQL是一个成熟的数据库查询语言，带有类似于连接、子查询、比较运算符、聚集、内置历史等的很多特性。

关于更多的信息，请访问Freebase开发者社区，特别是MQL参考指南<http://www.freebase.com/docs/mql>。开发者社区也为使用Freebase和MQL提供了很多应用，例如模式浏览器，用来查看Freebase中有哪些实体，这些实体有哪些属性，实体之间的关系：<http://schemas.freebaseapps.com/>。要写MQL查询，我们也推荐 MQL查询编辑器（<http://www.freebase.com/app/queryeditor/>），它提供了比浏览器地址栏更友好的界面。

21.5.2 使用Kettle 抽取Freebase数据

我们这里转换的例子是从Freebase数据库里抽取电影信息，转换名是freebase-films.ktr。例子可以从本书网站下载。图21-22是这个转换。

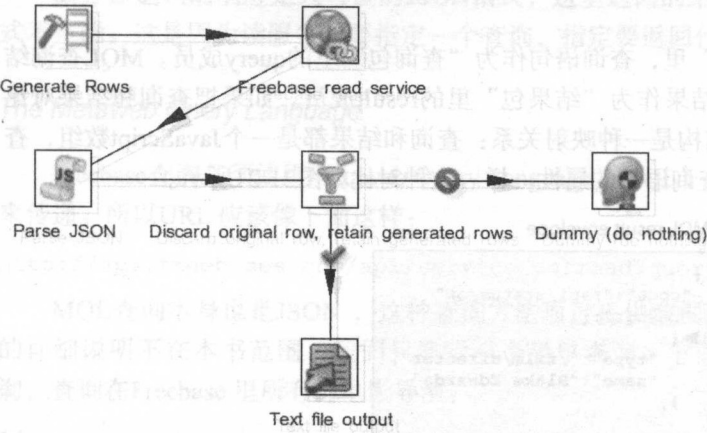


图21-22 Freebase电影转换

生成行

转换的第一个步骤是“生成行”步骤，该步骤的配置如图21-23所示。

该步骤生成以下几个常量字符串字段。

- url: Freebase 读服务的URL。
`http://api.freebase.com/api/service/mqlread`
- mql: MQL查询包内有一个查询语句，用来查询Freebase 里的电影。
`{"query":[{"type":"/film/film","name":null,"genre":[]}]}`
- name和genre: 这两个都是空，在转换的后面会被填上数据。

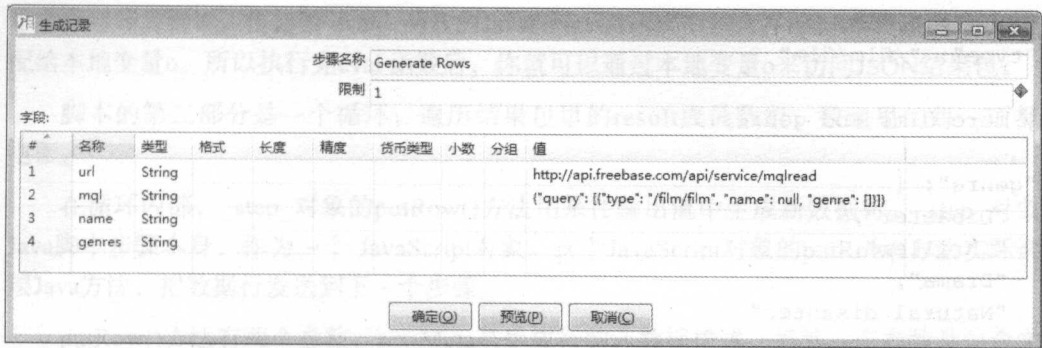


图21-23 “生成行”步骤

发出一个Freebase的读查询

第一个步骤生成的行被发送到“HTTP client”步骤，步骤名是“Freebase read service”。这个步骤的配置如图21-24所示。

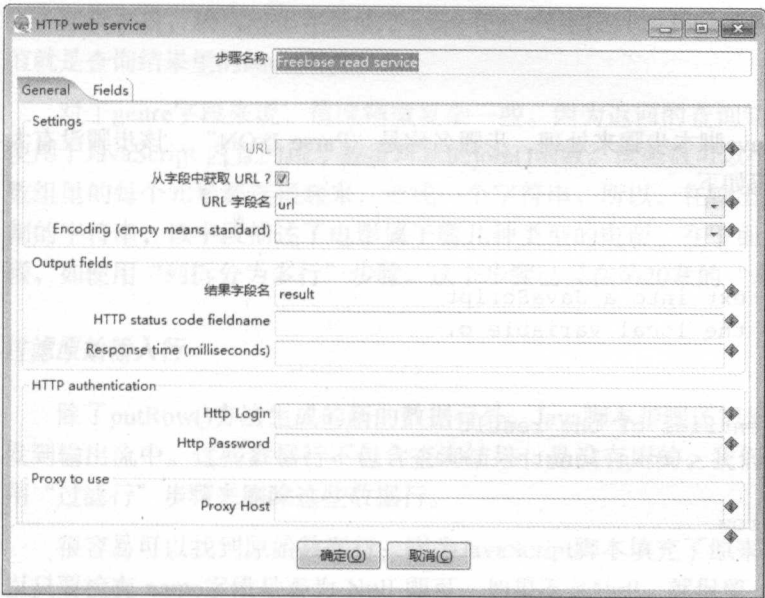


图21-24 Freebase的读服务

“HTTP client”步骤配置成从url字段接收一个URL。URL 返回的结果保存在输出流的result 字段里。为了把 MQL查询传递给Freebase的Web服务，配置窗口中的“参数”表格把输入流里的 mql字段映射到 query 参数，会从服务器得到下面的响应结果。

```
{
  "code": "/api/status/ok",
  "result": [
    {
      "genre": [
        "Black comedy",
        "Thriller",
        "Psychological thriller"
      ]
    }
  ],
}
```

```

    "name": ".45",
    "type": "/film/film"
  },
  ...many more films and genres...
  {
    "genre": [
      "Disaster",
      "Thriller",
      "Drama",
      "Natural disaster"
    ],
    "name": "10.5",
    "type": "/film/film"
  }
],
"status": "200 OK",
"transaction_id":
  "cache;cache04.p01.sjc1:8101;2010-04-06T22:23:15Z;0017"
}

```

处理Freebase 结果包

服务器返回的结果将使用JavaScript步骤来处理，步骤名称是“Parse JSON”。该步骤没有太多配置信息，主要是代码。代码如下：

```

var o, r, i;

//Turn response envelope text into a JavaScript
//object and assign it to the local variable o.
eval("o = " + result);

//look through the array entries of the result
for (i=0, rows = o.result; i < rows.length; i++){

  //Get current result row
  r = rows[i];

  //Use putRow() method of the builtin _step_
  //object to push a row to the output stream
  _step_.putRow(
    rowMeta, //use builtin row metadata
    [
      null, //field: url
      null, //field: mql
      r.name, //field: name
      r.genre.join(","), //field: genre
      null //field: result
    ]
  );
}

```

脚本首先使用JavaScript 内置的函数，eval()。该函数以字符串作为参数，动态地将输入字符

串参数解释为脚本。传递给 eval() 函数的JavaScript 表达式把包含在result字段里的JSON字符串分配给本地变量o。所以执行完eval 函数后,你可以通过本地变量o来访问JSON结果包。

脚本的第二部分是一个循环,遍历结果包里的result成员数组。数组里的每一项都是一个电影。

在循环内部, _step_ 对象的putRow()方法用来往输出流中生成新数据行。_step_ 对象实际是Java脚本步骤本身,作为一个 JavaScript对象。这个JavaScript对象的putRow()方法实际调用了底层Java方法,把数据行发送到下一个步骤。

putRow()方法有两个参数: rowMeta是生成行的元数据描述,另外一个参数是包含实际数据的数组。rowMeta对象直接使用了输入行的元数据,和 _step_ 对象一样, rowMeta对象也是内置的对象。为了避免增加字段的麻烦,我们提前在生成行步骤就定义了name和genre 两个空字段,这样在Java脚本步骤就可以直接拿输入字段输出字段。

说明: 要确保输出的元数据和输出的数据要保持一致,否则脚本不会工作,或者导致异常结果。

在脚本里,输出行里字段除了name和genre字段外,其余字段都被设置为null。name字段的值就是查询结果里的name属性。

对于genre字段来说,情况稍微复杂一些。因为返回的查询结果是一个数组。所以,脚本里使用了JavaScript 内置的用于数组对象的join()函数。该函数可以把数组对象串行化,使用逗号把数组里的每个元素都连接起来,形成一个字符串。所以,在输出流里, genre字段是一个逗号分割的字符串,该字段描述了电影属于哪几种类型的电影。在下面的步骤里可以继续处理genre字段,如使用“列拆分为多行”步骤。这个步骤已经在第20章的“多值属性”部分讨论过。

过滤原始输入行

除了putRow()方法生成的新的数据行外,Java脚本步骤还把从输入流中读取到的数据行也转发到输出流中。这些数据行不包含查询结果,是没有用的,我们要把这些数据行删除。可以使用“过滤行”步骤来删除这些数据行。

很容易可以找到原始数据行: 因为JavaScript脚本填充了原来为Null的name和genre字段,所以只要检查 name字段是否为 Null 即可。如果不是Null,就保留,如果是Null,就是原始的数据行,就要被丢弃到Dummy步骤。

文件排序

最后一个步骤是把数据行排序,再放到一个文件里。输出结果如下所示:

```
mql;names;genres;result
;.45;Black comedy,Thriller,Psychological thriller;
...many more rows...
;13 Ghosts;Horror;
;10.5;Disaster,Thriller,Drama,Natural disaster;
```

可以发现,转换完全去除了嵌套的JSON,把它转成了数据列。

21.6 RSS

RSS是Really Simple Syndication的缩写，是用于描述数据源更新状态的一组XML格式数据。在博客系统或新闻网站中使用得很多，它可以使访问者订阅或更新一个站点——被称为feed的一种应用（也常被称为XML-feed、RSS-feed，或webfeed）。

21.6.1 RSS结构

RSS是一种非常简单的XML格式，包含两个基本的元素，channel和items。下面我们分别讨论一下。

Channel

RSS文档顶层的元素是rss节点，带一个必需的版本属性，指定了RSS的版本号。我们下面使用的RSS版本号是2.0。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version = "2.0">
...
</rss>
```

rss节点下面有一个单一的channel节点，channel里包含了描述数据源的必要信息。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version = "2.0">
  <channel>
    <title>Press Releases</title>
    <link>
      http://www.pentaho.com/news/index.php?tab=news
    </link>
    <description>
      The latest Pentaho News
      and Press Releases from
      Pentaho.com.
    </description>
  </channel>
</rss>
```

channel节点下面包括下面几个节点。

- title: 频道的标题。
- link: 频道对应的URL 网址。
- description: 频道的描述信息。

除了上面几个节点外，还有几个可选的节点（如频道使用的语言、版权信息等），这些节点下面可能还有子节点。

条目

一个频道可能包含多个条目（item）节点。每个条目节点记录了对网站的一次变更。一般条

目节点里都会有description节点，保存本次变更的描述信息。另外还会有link节点，指向网站实际发生变化的部分。例如，在一个新网站的RSS内，条目代表了新的文章，description是文章的第一段或总结，link是这篇文章的URL。item节点内的所有节点，除了title和description外都是可选的。

一个条目可以有下面的节点。

- title: 条目的标题。
- description: 条目的描述。
- link: 条目的链接。
- pubData: 条目的发布日期。
- comments: 和条目相关的注释的URL。
- guid (Globally Unique Identifier): 字符串，是条目的唯一标识符。

下面是带有两个条目的channel的例子。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version = "2.0">
  <channel>
    <title>Press Releases</title>
    <link>
      http://www.pentaho.com/news/index.php?tab=news
    </link>
    <description>
      The latest Pentaho News
      and Press Releases from
      Pentaho.com.
    </description>
    <item>
      <title>
        Pentaho Releases Analyzer as First
        Deliverable in Agile BI Initiative
      </title>
      <link>
        http://www.pentaho.com/news/releases/20091104.php
      </link>
      <pubDate>Wed, 04 Nov 2009 9:00:00 EST</pubDate>
    </item>
    <item>
      <title>
        Pentaho Announces Strategic
        Technology Acquisition
      </title>
      <link>
        http://www.pentaho.com/news/releases/20091005.php
      </link>
      <pubDate>Mon, 05 Oct 2009 9:00:00 EST</pubDate>
    </item>
  </channel>
</rss>
```

另外还可以使用地理信息扩展RSS。Kettle 只支持基本地理度量 (GeoRSS-Simple) 和

GeoRSS GML上的点信息（经度和纬度）。下面例子演示了带基本地理信息的RSS 条目：

```
<georss:point>45.256 -71.92</georss:point >
```

下面例子是GML格式的RSS 条目：

```
<georss:where>
  <gml:Point>
    <gml:pos>45.256 -71.92</gml:pos>
  </gml:Point>
</georss:where>
```

21.6.2 Kettle对RSS的支持

对RSS 有基本了解后，继续看Kettle 如何处理RSS文件。Kettle使用RSS输入和RSS 输出步骤来读写 RSS文件。下面讨论这两个步骤。

RSS 输入步骤

RSS输入步骤位于设计器左侧面板树的“输入”类别下。RSS 输入步骤可以从大多数的RSS 格式和Atom syndication格式中抽取数据，支持的格式包括：

- RSS 0.90
- RSS 0.90 Netscape
- RSS 0.91 Userland
- RSS 0.92
- RSS 0.93
- RSS 0.94
- RSS 1.0
- RSS 2.0
- Atom 0.3
- Atom 1.0

这里我们讲一个例子，这个例子从Pentaho.com 网站读取发布于2010年1月份以后的所有文章。首先，要指定RSS所在的URL，如图21-25所示。

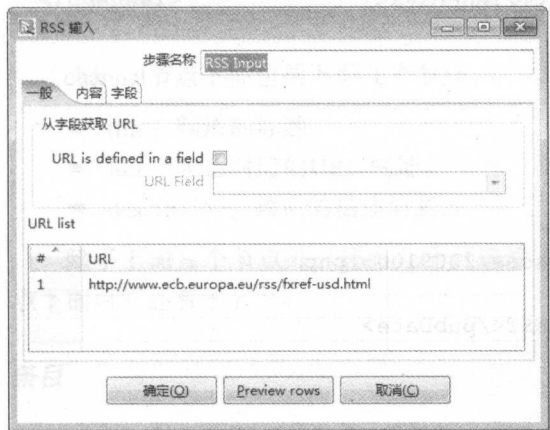


图21-25 RSS 输入步骤的“一般”标签

在一般标签里，可以输入静态的URL或者带变量的动态 URL。如果URL是在输入流的一个字段里，可以选中“URL is defined in a field”，同时在“URL Field”下拉列表中选择字段名。设置完URL，要在“内容”标签里设置只抽取2010年1月之后的更新，如图21-26所示。

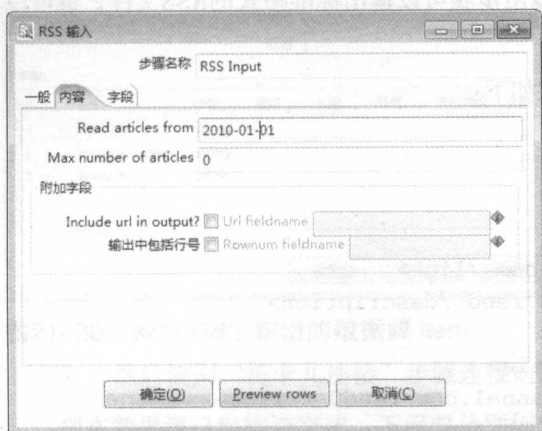


图21-26 RSS 输入步骤的“内容”标签

“内容”标签里有下面几个选项。

- **开始抽取日期：**输入一个日期，抽取该日期之后的文章。日期格式必须是yyyy-MM-dd HH:mm:ss。
- **最大文章数：**限制要抽取的文章数量（0代表抽取所有文章）。
- **输出URL（Include url in output）：**如果指定了一个静态URL，选中这个选项可以把这个URL添加到输出流里。“URL字段名”输入框里可以设置URL字段名。
- **输出中包括行号：**在很多输入步骤里都有这个选项，该选项将在输出字段里添加一个行号字段（第一行是1……）。“行号字段名”输入框里可以设置行号字段名。

最后一步是获取字段。在如图21-27的“字段”标签里使用“获取字段”按钮可以读取所有可用字段，可以移去或重命名这些字段。

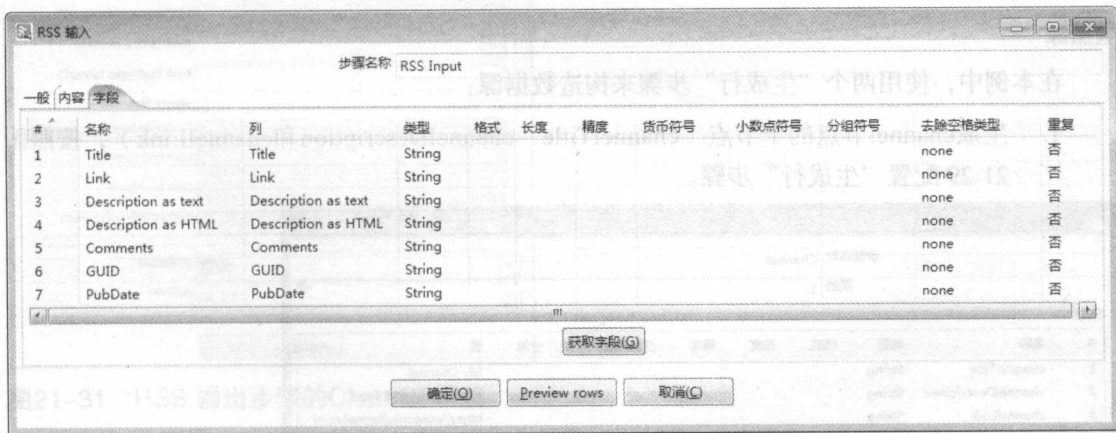


图21-27 选择字段

另外可以使用“预览行”（Preview rows）按钮来检查输入。可以看到，Kettle可以非常容易地从RSS中预览数据。

RSS 输出步骤

RSS 输出步骤和RSS 输入步骤功能相反。它把输入流的数据写成RSS格式的文件。这些文件随后通过Web服务提供给其他网站或应用。RSS 输出步骤可以输出标准格式的RSS文件，也可以通过配置发布自定义格式的RSS文件。

下面的例子演示了一条RSS的标准输出，格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>My Channel</title>
    <link>http://www.mychannel.com</link>
    <description>Example of RSS feed</description>
    <item>
      <title>Test item</title>
      <link>http://www.mychannel.com/feed0 </link>
      <description>An item in the channel</description>
      <guide>http://www.mychannel.com/feed0 </guide>
    </item>
  </channel>
</rss>
```

生成这个输出的转换如图21-28所示。

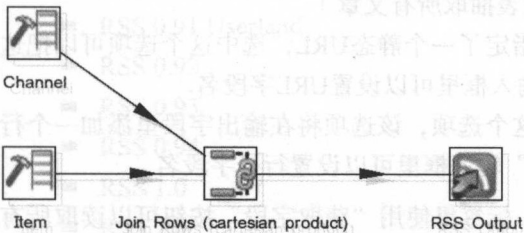


图21-28 演示RSS 输出的转换

数据源

在本例中，使用两个“生成行”步骤来构造数据源：

- 1. 生成channel节点的子节点（channelTitle、channelDescription和channelLink），按照图21-29 配置“生成行”步骤。

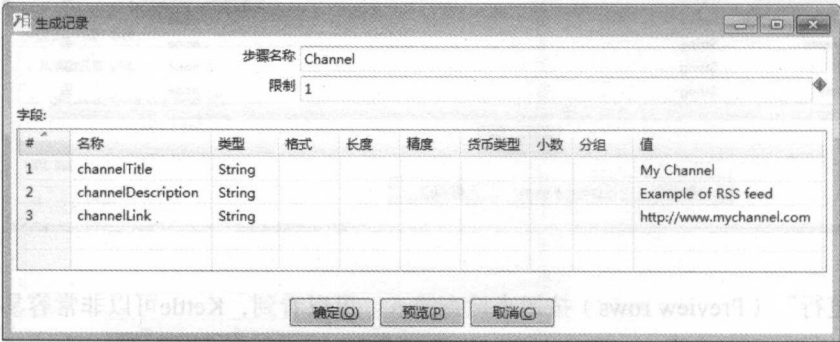


图21-29 构造 RSS 输出的数据源 channel

2. 生成item节点的子节点（itemTitle、itemDescription和itemLink），按照图21-30 配置“生成行”步骤。

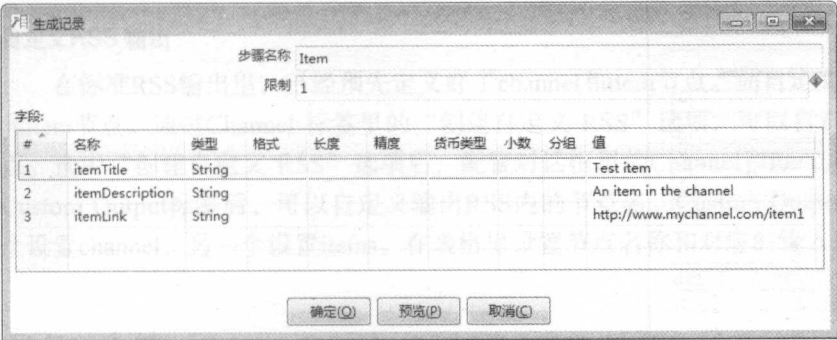


图21-30 构造 RSS 输出的数据源 item

3. 然后使用“笛卡儿连接”步骤连接这两个步骤。

现在数据源已经构造完成，下面要分别创建标准的和自定义的RSS 输出。

标准RSS 输出

标准RSS 输出里预定义了channel和item节点内必需的节点和可选的节点。要生成标准RSS 输出，不要选中“创建自定义RSS”（Create custom RSS）选项，如图21-31所示。

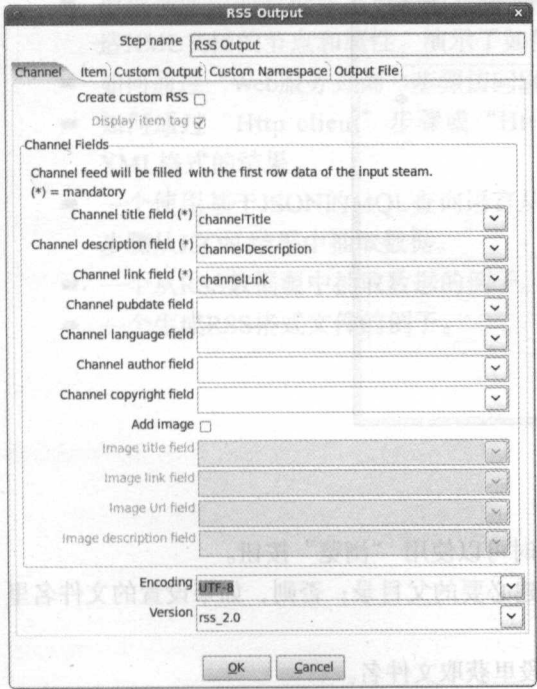


图21-31 RSS 输出步骤的Channel 标签

Channel 标签（如图21-31中所示）用来设置RSS里的channel节点；必填项字段前面标记着星号（*），其他都是选填项，选填项可以留空。在对话框下面，需要设置字符编码和RSS 版本。一般情况下要使用UTF-8 编码（UTF-8 也是XML的默认编码）。

将channel节点和输入流的字段映射完成后，需要对items节点做同样的工作。在图21-32中可

以看到，item节点下没有必须设置的节点，根据需要来设置。

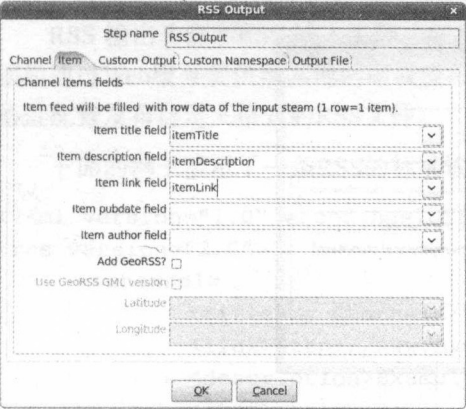


图21-32 RSS 输出步骤的Item 标签

最后的步骤是设置输出文件名，设置窗口如图21-33所示。

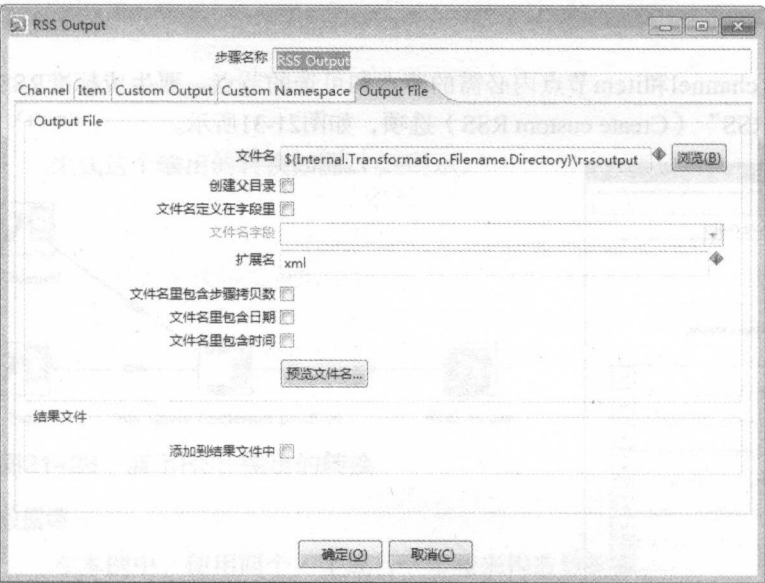


图21-33 RSS 输出步骤的Output File标签

在Output File标签里有下面几个选项可以设置。

- 文件名：指定 RSS 输出的全文件名。必要时可以使用“浏览”按钮。
- 创建父目录：如果选中，可以在运行时创建必要的父目录；否则，如果设置的文件名里有不存在的目录，PDI 将会执行失败。
- 文件名定义在字段里：如果选中，将从字段里获取文件名。
- 文件名字段：保存文件名的字段。
- 扩展名：添加到文件名后面的扩展名。
- 文件名里包含步骤拷贝数：在文件名后面添加步骤的拷贝数（拷贝数从0开始）。
- 文件名里包含日期：在文件名后面添加当前日期。
- 文件名里包含时间：在文件名后面添加当前时间。
- 预览文件名：预览设置好的文件名。

- 添加到结果文件中：将这个RSS 输出文件添加到结果文件中。

以上就是标准RSS 输出的配置，RSSOutputStandard.ktr例子演示了标准RSS 输出。

自定义RSS 输出

在标准RSS输出里，已经预先定义好了channel和item节点。而自定义RSS格式不需要channel和item节点。通过Channel 标签里的“创建自定义RSS”选项，可以自定义RSS 输出中的任何节点。选中“创建自定义RSS”选项后，配置对话框里的Channel和Item标签页将不再可用。选择Custom Output标签后，可以自定义输出RSS内的节点名。Custom Output标签内有两个表格，一个设置channel，另一个设置items。在表格里设置节点名称和对应的输入流里的字段。

21.7 小结

本章学习了Kettle如何处理Web和Web服务。

- Kettle访问Web的几种方式，例如HTTP步骤、SOAP、RSS和Apache的虚拟文件系统。
- Web 经常使用的几种数据格式，例如XML、HTML和JSON。
- 把XML导入到数据库的例子，演示了XSD步骤的使用方法，如何验证XML的有效性。
“XML输入步骤”如何使用XPath将XML中的数据加载到数据库。
- 把数据库中的数据导出成XML格式的例子，演示了如何通过“XML文件输入”步骤构造XML文档的节点和属性。演示了如何通过“XML连接”步骤生成嵌入的XML文档。
- 如何通过“Web服务查询”步骤访问SOAP格式的Web服务。
- 如何通过“Http client”步骤或“Http Post”步骤访问SOAP格式的Web服务，并解析XML格式的结果。
- 一个使用基于JSON的MQL查询语言从Freebase库中抽取数据的例子，如何使用Java脚本步骤从JSON 结果中抽取数据。
- 一个从RSS数据源中抽取数据的例子。
- 一个生成RSS格式文件的例子。

第22章 Kettle集成

Kettle包含许多数据整合的功能，用户通常把Kettle作为工具来使用。但是你也可以把Kettle当作开发库来使用，让它为你的软件 and 解决方案服务。软件复用是开源软件的主要特征，这样减少了研发新软件的时间。在本章，将介绍软件的授权，通过适当授权，软件才可以再次使用。还会介绍如何在你的Java 软件里集成Kettle，将使用Pentaho 软件栈里的其他软件为例来阐述如何集成Kettle。在每个例子中，将介绍Kettle的API集成方式。同时，还将阐述如何定制和参数化Spoon 界面。本章的最后部分将简单介绍OEM和二次发布。

22.1 Kettle API

在这一部分里，我们通过案例讲述API，让大家了解Kettle API的优势，同时介绍Kettle的LGPL协议。

22.1.1 LGPL协议

在Kettle研发团队决定把Kettle开源时，他们选择的开源协议是Lesser GUN Public License, 简称LGPL (<http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>)。该协议来自GNU，因为功能强大，FSF (Free Software Foundation) 把LGPL列为首选协议。LGPL协议允许Kettle作为商业（非开源）代码的链接库，使用Kettle的商业代码无须开源。但是如果你对Kettle源代码做了改变，并重新发布，LGPL协议要求你把这些改变也开源。如果你的代码只是使用Kettle，那你的代码不用开源，正好和GPL证书相反。GPL协议的软件也可以被使用或被嵌入，但是使用方也必须要求开源。

由此可见，LGPL的功能不仅在于可以使用Kettle API，还在于你能够销售自己的商业软件，这一点对软件零售商和软件公司都很重要，因此，我们说LGPL是对商业软件友好的授权。

LGPL给Kettle带来的另一个好处是：Kettle可以集成插件，而且插件的源代码可以不开源。这样任何公司都可以结合自身需要扩展Kettle，并且省时省力。关于这方面的更多内容，请参考第23章。过去的几年，很多公司都开发了不少步骤和作业输入插件。（参见<http://wiki.pentaho.com/display/EAI/List+of+Available+Pentaho+Data+Integration+Plug-Ins>。）有些插件的源代码是不公开的，但是也有很多插件的源代码是公开的，并且有些插件的代码并入了Kettle主分支，成为Kettle标准的步骤和作业项。所以LGPL是Kettle项目的最佳选择。

22.1.2 Kettle Java API

这一部分介绍如何获得Kettle源代码和编译源代码。同时也将讲解如何编写Java文档。最后，介绍Kettle的四个库里都包含哪些内容，以及通过API使用Kettle，都要引用哪些库。

源代码

有两种方式可以获得源代码。一是从sourceforge.net上下载，网址是<http://sourceforge.net/projects/pentaho>；二是从Subversion源代码资源库直接下载。Kettle源代码资源库以只读模式对所有用户开放，网址为<http://source.pentaho.org/svnkettleroot/Kettle>。

branches/下面是稳定的源代码。trunk/下面是最新的源代码。如果你想得到最新的源代码（但可能不太稳定），可以使用下面的命令从SVN检出最新分支（提示：UNIX下需要安装命令行svn才能运行。Windows兼容操作系统用户可以安装Tortoise SVN。参考<http://tortoisesvn.tigris.org/>）：

```
svn co http://source.pentaho.org/svnkettleroot/Kettle/trunk/
```

编译Kettle

找到源代码之后，就可以用Apache Ant 工具（<http://ant.apache.org>）编译Kettle。只需在Kettle源代码目录下的命令行上输入ant，就可以编译并发布到distrib/目录下。

编译javadoc

如果想查看源代码的Java文档，可以执行下面的命令：

```
ant javadoc
```

在docs/api 目录下就会生成javadoc，你会看到Kettle代码里类和方法的注释。使用浏览器查看帮助文件docs/api/index.html。

库和类路径

Pentaho Data Integration (Kettle)不仅仅是数据整合和商业智能工具，它也可以作为Java API来使用。API由以下四部分构成。

- Core：包括Kettle的核心类，保存在 kettle-core.jar文件中。

- Database: 包括Kettle数据库相关的类, 保存在 kettle-db.jar文件中。
- Engine: Kettle运行时类, 保存在kettle-engine.jar类中。
- GUI: 基于Eclipse SWT的图形界面相关的类, 例如Spoon, 保存在kettle-ui-swt.jar类中。

在下面的例子中, 需要把前三个 .jar文件放到类路径中, 连同Kettle的libext/目录下的.jar文件。你可能不需要所有这些文件, 但是在执行各种转换中, 难免会用上, 所以都复制过来更安全一些。

22.2 执行存在的转换和作业

接下来的部分介绍如何使用Java API运行已存在的转换和作业。

22.2.1 执行一个转换

当然, 以API方式执行Kettle转换的例子主要在Pentaho软件包中, 例如: Spoon、Pan、Carte, Pentaho BI 服务器以及Pentaho Data Integration服务器。用上面这些工具很容易执行一个转换。如果使用Kettle Java API执行转换也是非常容易的, 下面的代码, ExecuteTrans. Java, 只用了几行代码就执行了一个转换。

```
package example.ch22;

import org.pentaho.di.core.KettleEnvironment;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;

public class ExecuteTrans {
    public static void main(String[] args) throws Exception {
        String filename = args[0];

        KettleEnvironment.init();

        TransMeta transMeta = new TransMeta(filename);
        Trans trans = new Trans(transMeta);
        trans.prepareExecution(null);
        trans.startThreads();
        trans.waitUntilFinished();

        if (trans.getErrors() != 0) {
            System.out.println("Error encountered!");
        }
    }
}
```

这些代码都简单易懂。我们看这一行:

```
KettleEnvironment.init();
```

这个命令行设置了Kettle运行环境, 加载所有插件, 初始化日志环境和变量以及Kettle的home 目录。在初始化过程中, 发生了任何错误, 都会抛出异常。

下面这个命令行通过读取文件，创建了一个新的TransMeta（转换元数据）对象。

```
TransMeta transMeta = new TransMeta(filename);
```

通过Repository.loadTransformation()方法，也可以从资源库读取转换元数据。但是，使用资源库操作比较复杂。需要有资源库名称（ID），要输入用户名和密码才能访问资源库。资源的元数据都保存在Repositories.xml文件中。通过RepositoriesMeta.readData()方法可以读取这些元数据。有了这些资源库的连接信息，再通过PluginRegistry就可以获得资源库对象。

```
RepositoriesMeta repositoriesMeta = new RepositoriesMeta();
repositoriesMeta.readData();
RepositoryMeta repositoryMeta = findRepository( repositoryName );
PluginRegistry registry = PluginRegistry.getInstance();
Repository repository = registry.loadClass(
    RepositoryPluginType.class,
    repositoryMeta,
    Repository.class
);
repository.connect(username, password);
```

加载完转换元数据后，就可以执行下面的操作：

```
Trans trans = new Trans(transMeta);
trans.prepareExecution(null);
trans.startThreads();
trans.waitUntilFinished();
```

这部分代码创建了一个新的转换引擎对象，初始化转换，然后开始执行转换。因为转换是以多线程方式运行的，最后要等待所有的线程执行成功。

最后也是很重要的一步，要检查转换过程中是否有错误产生。

```
if (trans.getErrors() != 0) {
    System.out.println("Error encountered!");
}
```

转换过程中出现的任何错误都会出现在控制台上。

22.2.2 执行一个作业

和执行转换相同，执行作业的软件也大部分在Pentaho软件包中，例如：Spoon、Kitchen、Carte、Pentaho BI 服务器以及Pentaho Data Integration服务器。执行作业的代码也和执行转换的代码类似，如下所示（ExecuteJob.java）：

```
package example.ch22;

import org.pentaho.di.core.KettleEnvironment;
import org.pentaho.di.job.Job;
import org.pentaho.di.job.JobMeta;

public class ExecuteJob {
    public static void main(String[] args) throws Exception {
        String filename = args[0];
```

```
KettleEnvironment.init();

JobMeta jobMeta = new JobMeta(filename, null);
Job job = new Job(null, jobMeta);
job.start();
job.waitUntilFinished();

if (job.getErrors() != 0) {
    System.out.println("Error encountered!");
}
```

从文件里加载作业元数据非常容易。这段代码和前面转换的例子类似，唯一不同的是JobMeta构造函数里有一个值为null的资源库引用（因为没用资源库）。

22.3 应用程序中嵌入Kettle

Pentaho的软件栈里有一百多个大大小小的组件。许多组件都是小的内核通用组件，其他的一些组件是大的软件集合，比如BI服务器，有几百兆。Pentaho内核通用组件的目的就是最大限度地实现代码复用，这样可以减少研发的人力。Kettle位于软件栈的中间位置，它使用Pentaho的一些通用组件，但本身也被很多其他组件使用。所以说，在各类应用中集成Kettle，会非常简单。在下面的章节里，将介绍一些集成Kettle的案例和源代码，帮助你在应用中集成Kettle。

22.3.1 Pentaho 报表

Pentaho 报表可以从Kettle转换的任何步骤中获取数据。因为Pentaho报表中也嵌入了Kettle库。Pentaho 报表设计器（Pentaho Report Designer, PRD）的数据源菜单中有PDI 类型的数据源，可以创建一个PDI 类型的数据源。选中PDI 类型的数据源后，会打开一个对话框，在对话框里选择转换和作为数据源的步骤。Pentaho 报表设计器如图22-1所示。

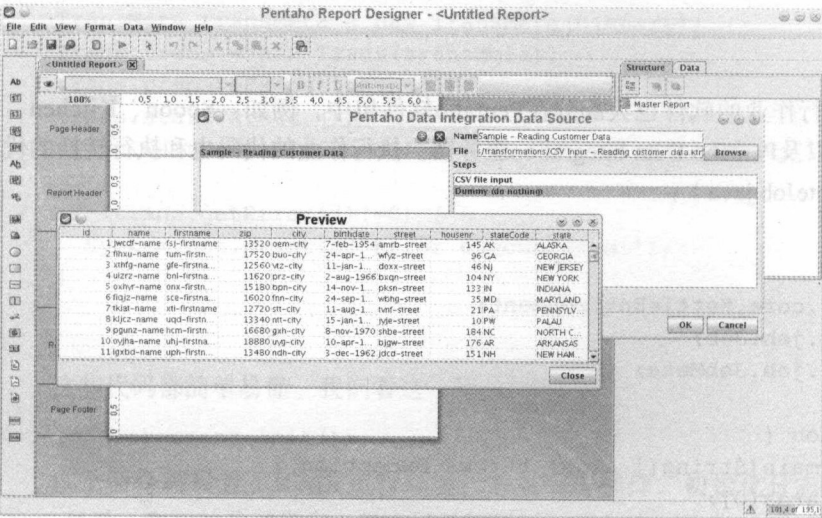


图22-1 使用Pentaho 报表设计器

说明：因为Kettle和Pentaho 报表的发布周期不同，而且这两个软件经常变化，所以要注意版本是否搭配。如Pentaho Report 3.5可以访问Kettle3.2，Pentaho Report 3.6（或以后版本）可以访问Kettle 4.0。

从Kettle转换的步骤里读取数据非常容易。我们已经知道了怎样执行转换，可以把例子扩展一下，让例子可以从某个步骤读取数据，如下面的代码（ReadFromStep.java）：

```
package example.ch22;

import java.util.ArrayList;
import java.util.List;

import org.pentaho.di.core.KettleEnvironment;
import org.pentaho.di.core.RowMetaAndData;
import org.pentaho.di.core.exception.KettleStepException;
import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.RowAdapter;
import org.pentaho.di.trans.step.RowListener;
import org.pentaho.di.trans.step.StepInterface;

public class ReadFromStep {
    public static void main(String[] args) throws Exception {
        String filename = args[0];
        String stepname = args[1];

        KettleEnvironment.init();

        TransMeta transMeta = new TransMeta(filename);
        Trans trans = new Trans(transMeta);
        trans.prepareExecution(null);

        final List<RowMetaAndData> rows = new ArrayList<RowMetaAndData>();
        RowListener rowListener = new RowAdapter() {
            public void rowWrittenEvent(RowMetaInterface rowMeta, Object[] row)
                throws KettleStepException {
                rows.add(new RowMetaAndData(rowMeta, row));
            }
        };
        StepInterface stepInterface = trans.findRunThread(stepname);
        stepInterface.addRowListener(rowListener);

        trans.startThreads();
        trans.waitUntilFinished();

        if (trans.getErrors() != 0) {
            System.out.println("Error");
        } else {
            System.out.println("We read"+rows.size()+"rows from step"+
```

```

        stepname);
    }
}

```

和ExecuteTrans.java例子唯一不同的是, 这里需要创建一个行监听对象RowListener, 并把这个监听对象注册到某个步骤拷贝上。这个例子只有一个步骤拷贝。如果想从某个特定的步骤拷贝获取数据就要使用getRunThread(stepname, copy)方法。

当注册了RowListener监听对象, 当有行读取和写入时, 或有数据行写入到错误处理步骤时, 这些事件都可以被监听到。在转换运行的过程中, 事件通知就同步进行了。这样你就可以通过数据流的方式来处理数据。在这个例子里, 我们把所有输出行写到一个Java 列表里, 等转换运行完就可以处理这个Java列表。

22.3.2 把数据放到转换里

现在我们了解了如何使用RowListener接口以数据流的方式从步骤里读取数据, 下面我们再看如何把数据放到转换里。

使用Result对象传递数据

可以使用“复制行到结果”和“从结果获取行”步骤在作业的转换之间传递数据。这种方法实际上利用了内存中的缓存, 不是基于数据流的方式。也就是说使用Result对象的这种方法不适合于大数据量的情况。可以使用Java API来实现“复制行到结果”步骤的功能。在这个例子里, 我们把前面的代码改一下, 让它包括下面的三行代码, 以及相关的创建数据行的方法(见PassDataToTransfer.java):

```

...
Result result = new Result();
result.setRows(createRows());
transMeta.setPreviousResult(result);
...

private static List<RowMetaAndData> createRows() {
    List<RowMetaAndData> list = new ArrayList<RowMetaAndData>();

    RowMetaAndData one = new RowMetaAndData();
    one.addValue("string", ValueMetaInterface.TYPE_STRING,
        "A sample String");
    one.addValue("date", ValueMetaInterface.TYPE_DATE, new Date());
    one.addValue("number", ValueMetaInterface.TYPE_NUMBER,
        Double.valueOf(123.456));
    one.addValue("integer", ValueMetaInterface.TYPE_INTEGER,
        Long.valueOf(123456L));
    one.addValue("big_number", ValueMetaInterface.TYPE_BIGNUMBER,
        new BigDecimal("1234593943942.39430243953243239434"));
    one.addValue("boolean", ValueMetaInterface.TYPE_BOOLEAN,
        Boolean.TRUE);
    one.addValue("binary", ValueMetaInterface.TYPE_BINARY,
        new byte[] { 0x44, 0x50, 0x49, } );
}

```



```
list.add(one);

return list;
}
```

在代码里，我们把一组数据行传给了Result对象，并把Result对象设置为当前转换的前一个转换的Result。这个例子也显示了Kettle里的不同数据类型。关于Kettle数据行的更多内容请参考第23章。

现在只要在转换里设置一个“从结果获取数据行”步骤，就可以接收其他程序传来的数据行了。

流方式传递数据

如果要传递大量的数据，而且还不能占用太多的内存，就需要使用流的方式把数据传入转换。这时就要使用“记录注射 (Injector)”步骤。这个步骤可以在转换运行时接收数据。为了能把数据传给这个步骤，还要给Trans对象添加一个RowProducer对象。转换流程如图22-2所示。

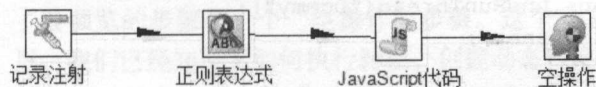


图22-2 使用“记录注射”步骤

在这个转换里，没有从任何数据源读取数据。转换通过“记录注射”步骤在运行时从外部接收数据，接着做正则表达式匹配和JavaScript脚本步骤，最后把数据发送给空操作步骤。下面的代码演示了如何把数据传递给“记录注射”步骤（InjectDataIntoTransformation.java）：

```
package example.ch22;
```

```
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

```
import org.pentaho.di.core.KettleEnvironment;
import org.pentaho.di.core.Result;
import org.pentaho.di.core.RowMetaAndData;
import org.pentaho.di.core.exception.KettleStepException;
import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.core.row.ValueMetaInterface;
import org.pentaho.di.trans.RowProducer;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.RowAdapter;
import org.pentaho.di.trans.step.RowListener;
import org.pentaho.di.trans.step.StepInterface;
```

```
public class InjectDataIntoTransformation {
    public static void main(String[] args) throws Exception {
        String filename = args[0];
```

```

KettleEnvironment.init();

TransMeta transMeta = new TransMeta(filename);

Result result = new Result();
result.setRows(createRows());
transMeta.setPreviousResult(result);

Trans trans = new Trans(transMeta);
trans.prepareExecution(null);

final List<RowMetaAndData> rows = new ArrayList<RowMetaAndData>();
RowListener rowListener = new RowAdapter() {
    public void rowWrittenEvent(RowMetaInterface rowMeta, Object[] row
        ) throws KettleStepException {
        rows.add(new RowMetaAndData(rowMeta, row));
    }
};
StepInterface stepInterface = trans.findRunThread("Dummy");
stepInterface.addRowListener(rowListener);

RowProducer rowProducer = trans.addRowProducer("Injector", 0);

trans.startThreads();

for (RowMetaAndData row : createRows() ) {
    rowProducer.putRow(row.getRowMeta(), row.getData());
}
rowProducer.finished();

trans.waitUntilFinished();

if (trans.getErrors()!=0) {
    System.out.println("Error");
} else {
    System.out.println("We got back"+rows.size()+"rows");
}
}

private static List<RowMetaAndData> createRows() {
    //see the previous example
}
}

```

在转换开始之前，让转换创建一个RowProducer对象非常重要。一旦线程启动，只有通过putRow()方法，把数据传给“记录注射”步骤，“记录注射”步骤才能工作。注意当步骤的行集（RowSet）缓存满了，步骤就会停止运行。默认的行集缓存是10 000行（请参考第15章）。如果注射的速度大于数据处理的速度，“记录注射”步骤会暂时停止执行。如果注射完了，就可以调用finished()方法来告诉“记录注射”步骤数据已经注射完毕。最后调用转换的waitUntilFinished方法，等待其他步骤都执行完。

设置参数和变量

通过参数也可以把信息传递给转换或作业。在转换里可以通过变量或参数的形式来实现。

使用Java API设置变量非常容易；在创建trans对象之前，调用transMeta的设置变量方法即可。

```
transMeta.setVariable("VARIABLE_NAME", "value");
```

通过类似的方法可以设置参数：

```
transMeta.setParameterValue("PARAMETER_NAME", "value");
```

注意，如果参数名不存在，这个方法会抛出异常。使用transMeta.listParameters()方法可以列出转换里的所有参数。使用transMeta.getParameterDefault()还可以获得每个参数的默认值。

在JobMeta类里也有一样的设置变量和设置参数的方法。

22.3.3 动态转换

当你在Spoon里单击某个步骤的“预览”按钮，实际就动态地创建了一个转换。转换包括了要预览的步骤和一个“空操作”步骤。这个动态生成的转换被执行，执行结果显示在对话框里。我们已经知道了如何执行转换，创建动态转换也并不困难。下面看一个例子。已知一个CSV文件的文件名和文件格式，希望Kettle能读取这个文件，文件的分隔符是逗号，封闭符是双引号。输出写到“空操作”步骤。这个准备动态生成的转换如图22-3所示。

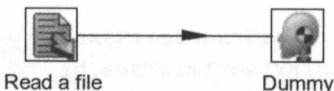


图22-3 动态生成的预览转换

如果CSV文件的名字和格式事先并不知道，也可以在运行时从其他地方获得。也就是说步骤的元数据可以存储在其他地方。下面的CsvFileReader类可以在运行时动态获得这些元数据（见第22章的CsvFileReader.java文件）。

```
public class CsvFileReader {

    private static String STEP_READ_A_FILE = "Read a file";
    private static String STEP_DUMMY = "Dummy";

    private String filename;
    private TextFileInputField[] inputFields;
    private List<RowMetaAndData> rows;

    public CsvFileReader(String filename,
                        TextFileInputField[] inputFields) {
        this.filename = filename;
        this.inputFields = inputFields;
    }

    public void read() throws Exception {
        KettleEnvironment.init();
```

```

// Create a new transformation...
//
TransMeta transMeta = new TransMeta();
transMeta.setName("sample04");

// Create the step to read the file
//
CsvInputMeta inputMeta = new CsvInputMeta();
inputMeta.setDefault(); // comma separated, " enclosed with header.
inputMeta.setFilename(filename);
inputMeta.setInputFields(intFields);

StepMeta inputStep = new StepMeta(STEP_READ_A_FILE, inputMeta);
inputStep.setLocation(50, 50);
inputStep.setDraw(true);
transMeta.addStep(inputStep);

// Create the dummy place-holder step
//
DummyTransMeta dummyMeta = new DummyTransMeta();
StepMeta dummyStep = new StepMeta(STEP_DUMMY, dummyMeta);
dummyStep.setLocation(150, 50);
dummyStep.setDraw(true);
transMeta.addStep(dummyStep);

// Create a hop between the 2 steps
//
TransHopMeta hop = new TransHopMeta(inputStep, dummyStep);
transMeta.addTransHop(hop);

// Now we can execute the transformation
//
Trans trans = new Trans(transMeta);
trans.prepareExecution(null);

rows = new ArrayList<RowMetaAndData>();
RowListener rowListener = new RowAdapter() {
    public void rowWrittenEvent(RowMetaInterface rowMeta,
                                Object[] row
                                ) throws KettleStepException {
        rows.add(new RowMetaAndData(rowMeta, row));
    }
};

StepInterface stepInterface = trans.findRunThread(STEP_DUMMY);
stepInterface.addRowListener(rowListener);

trans.startThreads();
trans.waitUntilFinished();

if (trans.getErrors() != 0) {
    System.out.println("Error");
}

```



```

    } else {
        System.out.println("We read "+rows.size()+" rows from step "
            +STEP_DUMMY);
    }
}

public List<RowMetaAndData> getRows() {
    return rows;
}

```

从上面的代码可以看出来，我们不用再从文件或资源库加载转换，转换从一开始就用代码创建。可以把步骤、数据库连接、步骤连接等对象都加入到 `TransMeta` 类里。不用担心，因为在运行时才开始解析元数据。在这个例子里，我们添加了两个步骤和一个步骤连接：

```

TransMeta transMeta = new TransMeta();
...
transMeta.addStep(inputStep);
...
transMeta.addStep(dummyStep);
...
transMeta.addTransHop(hop);

```

这和把两个步骤拖到画布上，并把这两个步骤连接起来的效果是一样的。

这里有一些使用 `CsvFileReader` 类的代码：

```

TextFileInputField id = new TextFileInputField("id", -1, 8);
id.setTrimType(ValueMetaInterface.TRIM_TYPE_BOTH);
id.setFormat("#");
id.setType(ValueMetaInterface.TYPE_INTEGER);

TextFileInputField firstname = new TextFileInputField("firstname",
    -1, 50);
firstname.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField name = new TextFileInputField("name", -1, 50);
name.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField zip = new TextFileInputField("zip", -1, 15);
zip.setTrimType(ValueMetaInterface.TRIM_TYPE_LEFT);
zip.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField city = new TextFileInputField("city", -1, 50);
city.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField birthdate = new TextFileInputField("birthdate",
    -1, -1);
birthdate.setFormat("yyyy/MM/dd");
birthdate.setType(ValueMetaInterface.TYPE_DATE);

TextFileInputField street = new TextFileInputField("street", -1,
    50);
street.setType(ValueMetaInterface.TYPE_STRING);

```

```

TextFileInputField housenr = new TextFileInputField("housenr", -1, 2, 15);
housenr.setTrimType(ValueMetaInterface.TRIM_TYPE_LEFT);
housenr.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField statecode = new TextFileInputField("statecode", -1, 2);
statecode.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField state = new TextFileInputField("state", -1, 50);
state.setType(ValueMetaInterface.TYPE_STRING);

TextFileInputField[] inputFields = new TextFileInputField[] {
    id, firstname, name, zip, city, birthdate, street, housenr,
    statecode, state,
};

String filename = "samples/transformations/files/customers-100.txt";
CsvFileReader reader = new CsvFileReader(filename, inputFields);
reader.read();

```

22.3.4 动态模板

如果能通过代码把转换保存到一个文件，这个文件就可以通过Spoon打开和编辑了。调用getXML()方法可以把转换对象序列化为XML格式的字符串，然后转换就可以保存到文件里：

```

String xml = XMLHandler.getXMLHeader() + transMeta.getXML();
DataOutputStream dos = new DataOutputStream(
    KettleVFS.getOutputStream("csv-reader.ktr", false)
);
dos.write(xml.getBytes(Const.XML_ENCODING));
dos.close();

```

如果你要读取几百个不同类型的文件，你可能要为每个类型的文件都创建一个转换，转换里包含一个“CSV 文件输入”步骤，一个“表输出”步骤，一个错误处理类步骤。如果能创建一个程序，由这个程序设置不同类型文件的元数据，并生成转换，可能会更有效。CsvFileReader类就成为一个动态的模板。其次你也可以根据不同的属性生成几百个不同的转换，根据需求来运行。

大多数ETL工具（不是全部），都需要事先知道步骤的输出。所以，从传统方法来看，需要在Spoon里手工创建这几百个转换，并一个一个地配置这些转换里的步骤。通过使用CsvFileReader例子中的模板就可以解决这样的问题。是否要把这些转换保存在文件里并整合到你的ETL 框架里，可以完全取决于你，因为这些转换可以在运行时构造和执行。也可以把动态构造和执行转换的代码封装成步骤或作业项的插件，关于这部分内容请参考第23章。

22.3.5 动态作业

Spoon 里的“复制表”向导就是动态生成作业的一个例子。这个向导可以动态生成一个作

业，这个作业把一个数据库里的表复制到另一个数据库里。这个生成的作业为每个选中的表都生成一个“执行SQL脚本”作业项和一个把数据加载到目标表的转换作业项。

Kettle论坛里经常有动态生成和执行SQL脚本的问题，下面的代码（DynamicJob.java）把转换文件名作为参数来生成作业，作业会执行必要的SQL。因为可能会有零个、一个或多个数据库，所以在生成的作业里也可能会有零个、一个或多个SQL脚本作业项。

```
package example.ch22;

import java.util.HashSet;
import java.util.List;

import org.pentaho.di.core.ObjectLocationSpecificationMethod;
import org.pentaho.di.core.SQLStatement;
import org.pentaho.di.core.database.DatabaseMeta;
import org.pentaho.di.job.JobHopMeta;
import org.pentaho.di.job.JobMeta;
import org.pentaho.di.job.entries.sql.JobEntrySQL;
import org.pentaho.di.job.entries.trans.JobEntryTrans;
import org.pentaho.di.job.entry.JobEntryCopy;
import org.pentaho.di.trans.TransMeta;

public class DynamicJob {

    public static JobMeta generateJobMeta(String transFilename
    ) throws Exception {
        JobMeta jobMeta = new JobMeta();
        jobMeta.setName("sample05");

        int x = 50;
        int y = 50;

        // Add the start entry...
        //
        JobEntryCopy startCopy = JobMeta.createStartEntry();
        startCopy.setLocation(x, y);
        startCopy.setDrawn();
        jobMeta.addJobEntry(startCopy);
        JobEntryCopy lastCopy = startCopy;

        // Determine the SQL and databases needed to
        // execute the transformation
        //
        TransMeta transMeta = new TransMeta(transFilename);
        HashSet<DatabaseMeta> databases = new HashSet<DatabaseMeta>();
        List<SQLStatement> sqlStatements = transMeta.getSQLStatements();
        for (SQLStatement stat : sqlStatements) {
            databases.add(stat.getDatabase());
        }
    }
}
```

```

// Add "Execute SQL script" for every used database...
//
for (DatabaseMeta databaseMeta : databases) {
    JobEntrySQL jobEntrySql = new JobEntrySQL();
    jobEntrySql.setDatabase(databaseMeta);

    String sql = "";
    for (SQLStatement sqlStatement : sqlStatements) {
        if (sqlStatement.getDatabase().equals(databaseMeta)) {
            if (!sqlStatement.hasError() && sqlStatement.hasSQL()) {
                sql += sqlStatement.getSQL();
            }
        }
    }
    jobEntrySql.setSQL(sql);

    JobEntryCopy sqlCopy = new JobEntryCopy(jobEntrySql);
    sqlCopy.setName("SQL for "+databaseMeta.getName());
    x+=100;
    sqlCopy.setLocation(x, y);
    sqlCopy.setDrawn();
    jobMeta.addJobEntry(sqlCopy);

    JobHopMeta sqlHop = new JobHopMeta(lastCopy, sqlCopy);
    jobMeta.addJobHop(sqlHop);
    lastCopy = sqlCopy;
}

// Now execute the transformation as well...
//
JobEntryTrans jobEntryTrans = new JobEntryTrans();
jobEntryTrans.setSpecificationMethod(
    ObjectLocationSpecificationMethod.FILENAME);
jobEntryTrans.setFileName(transFilename);

JobEntryCopy transCopy = new JobEntryCopy(jobEntryTrans);
transCopy.setName("Execute "+transFilename);
x+=100;
transCopy.setLocation(x, y);
transCopy.setDrawn();
jobMeta.addJobEntry(transCopy);

JobHopMeta transHop = new JobHopMeta(lastCopy, transCopy);
jobMeta.addJobHop(transHop);
lastCopy = transCopy;

return jobMeta;
}
}

```

从TransMeta.getSQLStatements()方法里可以看出, 生成一个转换里所有的SQL语句非常容

易。注意只有非动态转换才能生成SQL语句。例如，你在“表输出”步骤里定义了一个表名，这个表名是一个动态设置的变量。这时就不可能获得转换里所有的SQL 语句。

generateJobMeta()方法最后生成一个JobMeta对象，这个JobMeta对象可以执行也可以保存到磁盘上。同样，生成的作业是否要保存为XML格式的文件（或资源库），或者直接执行，或者做成作业项插件，也同样取决于你。

22.3.6 在Kettle里执行动态ETL

当然可以在“用户自定义Java类”和“Java脚本”步骤里执行刚才的代码。但对于这个例子，我们用下面的方法扩展DynamicJob就可以执行作业了。

```
public static Result executeTransformation(String transFilename
) throws Exception {
    JobMeta jobMeta = generateJobMeta(transFilename);
    Job job = new Job(null, jobMeta);
    job.start();
    job.waitUntilFinished();
    return job.getResult();
}
```

这个方法会执行指定的转换。现在就可以把这个类放到一个jar文件里，并把这个jar 文件放到Kettle发布目录的libext/目录下。这样Kettle就可以使用这个类了。

下面就可以使用一些 JavaScript代码来执行一组转换（.ktr）文件了。通过“获取文件名步骤”获得一组转换文件名。把这些文件名传给“Java脚本”步骤，Java脚本步骤里的代码也很简单：

```
var result = Packages.example.ch22.DynamicJob
    .executeTransformation(filename);
var errors = result.getNrErrors();
```

这样，这组转换就可以动态执行了，转换依赖的SQL会在转换执行之前被执行。

22.3.7 Result

前面的“Java脚本”例子里，执行作业会返回一个Result对象。这个对象不但包含了错误数还包含了很多其他信息，这些信息用来在转换和作业之间传递。表22-1显示了Result对象里包含的各种信息。表里包含了获取这些信息的方法、数据类型、描述。

表22-1 Result对象

| 方法 | 数据类型 | 描述 |
|---------------|---------|------------------------------------|
| getResult | boolean | 如果对象（转换或作业）执行成功返回true，如果有错误返回false |
| getExitStatus | int | shell 脚本作业项的退出状态 |
| getEntryNr | int | 执行完的作业项个数，每执行完一个作业项，就会增加一个 |
| getNrErrors | long | 错误的个数 |

续表

| 方法 | 数据类型 | 描述 |
|---------------------|--------------------------|----------------------------------|
| getNrLinesInput | long | 从文件或数据库里读取到的数据行数 ^a |
| getNrLinesOutput | long | 写入到文件或数据库里的数据行数 ^a |
| getNrLinesRead | long | 从前一个步骤里读取到的数据行数 ^a |
| getNrLinesUpdated | long | 文件或数据库里更新的数据行数 ^a |
| getNrLinesWritten | long | 写入到下一个步骤里的数据行数 ^a |
| getNrLinesDeleted | long | 从文件或数据库里删除的数据行数 ^a |
| getNrLinesRejected | long | 被文件或数据库拒绝的数据行数 ^a |
| getRows | List <RowMetaAndData> | 结果数据行 |
| isStopped | boolean | 作业是否是手工停止 |
| getResultFilesList | List <ResultFile> | 结果文件列表（转换/作业执行中用到的文件） |
| getNrFilesRetrieved | int | 从FTP、SFTP获取的文件个数 |
| getLogText | String | 作业/转换执行过程的日志 |
| getLogChannelId | String | 作业/转换的日志通道ID。用来在不同日志表里跟踪同一次执行的日志 |

<http://wiki.pentaho.com/display/EAI/Evaluating+conditions+in+The+JavaScript+job+entry>

a. 需要在“转换设置”对话框的“日志”标签下选择一个能代表这个度量值的步骤。

22.3.8 替换元数据

有时需要加载一个已有的作业/转换，但是需要在运行时替换数据库或一些步骤的设置。尽管可以使用参数或变量，但有时也要增加新的对象。

我们这里要讨论的是如何替换一个转换里已有的数据库连接。我们不但可以变更主机名、用户名等参数，甚至可以变更数据库类型。变更数据库类型不能使用变量或参数。在这个例子里，转换里的数据库连接名称是DB，它用在多个步骤里。

第一步是创建一个新的DatabaseMeta对象。如果要替换其他的对象，如步骤、作业项、集群子服务器、分区模式和集群模式等，也都有相应的创建方法：

```
DatabaseMeta databaseMeta = new DatabaseMeta(
"DB", "MySQL", "JDBC", "localhost",
"test", "3306", "user", "password"
);
```

直接使用API替换

使用addOrReplaceDatabase()方法，替换转换或作业中名称为“DB”的数据库连接：

```
transMeta.addOrReplaceDatabase(databaseMeta);
```

已有数据库可能被多个步骤引用，这个方法将修改已有数据库的属性，而不影响其他步骤对数据库的引用。

使用共享对象文件

共享对象文件里包含了数据库对象、步骤和其他可以被复用的对象。在 Spoon 窗口左侧的对象树上，右键单击某个对象，在弹出的右键菜单上选择“共享”（Share）。这个对象就会被保存在一个共享文件中。默认的文件是\$KETTLE_HOME/.kettle/shared.xml。

共享文件里对象的优先级高于转换/作业里的同名对象。所以可以使用共享对象来替代转换/作业里的同名对象，如替代同名的数据库连接对象。但是需要先创建共享对象文件：

```
SharedObjects sharedObjects = new SharedObjects();
sharedObjects.storeObject(databaseMeta);
sharedObjects.setFilename("/tmp/shared.xml");
sharedObjects.saveToFile();
```

然后让TransMeta对象加载共享文件：

```
transMeta.setSharedObjectsFile("/tmp/shared.xml");
transMeta.readSharedObjects();
```

如果你想为所有的作业设置默认的共享对象文件，可以用下面的语句：

```
System.setProperty(Const.KETTLE_SHARED_OBJECTS, "/tmp/shared.xml");
```

同时，注意在你的转换/作业里不能有其他共享对象文件目录，上面的设置才能生效。

22.4 OEM版本和二次发布版本

本节我们介绍如何定制化Kettle和Spoon的用户界面，使它们能更符合需要。我们也要解释如何再把Kettle进行二次发布，以及目前存在着哪些Kettle的二次发布版本。

22.4.1 创建PDI的OEM版本

Kettle提供了插件和参数化的功能，所以扩展Kettle的功能非常容易。但是，有的公司或组织想创建一个自定义名称的版本或OEM的版本，以便把Kettle嵌入到一个更大的业务系统里。只要你不修改Kettle源代码里的任何内容，根据LGPL协议这一切都是可以的。这也是为什么我们提供了一些功能，可以用来设置Kettle尤其是Spoon的界面感观的原因。

Kettle的大部分代码支持国际化，国际化也称为i18n（i-18characters-n）。Spoon界面里的每块文字，都被翻译成了多国语言。被存储在不同的资源文件中，每块文字都有唯一的键值。例如Spoon这个文字就存储在下面的文件里，键值是Spoon.Application.Name：

```
src-ui/org/pentaho/di/ui/spoon/messages/messages_en_US.properties
```

通过修改上面的文件就可以修改Spoon的名字。但这就是二次发布了，因为修改了Kettle的代码。如果从OEM的角度，显然不希望这么做，因为这需要用户自己去维护Kettle代码。为了解决这个问题，Kettle推出了界面感观（Look and Feel, LAF）管理器，通过这个管理器，可以脱离Kettle的代码，自己定义i18n的键、图片和属性文件。这些配置工作都可以通过自定义的属性文件完成。例如：

```
com/example/book/ui/spoon/messages/messages_en_US.properties
```

可以把所有要修改的Spoon的键值放在这个文件里, 例如下面的键值:

```
Spoon.Application.Name=Data Integration Designer
```

再使用下面的命令把资源文件打包到一个jar 文件里。

```
jar -cvf customizations.jar com/
```

然后把 customizations.jar 文件放在libext/目录下, 也就是Kettle的class路径。剩下要做的就是告诉 Kettle从哪里去找自定义的资源文件。这时需要设置 ui/laf.properties文件里的LAFpackage属性:

```
LAFpackage=com.example.book
```

下次再启动Spoon, 就会发现窗口的标题“Spoon” 被其他名称代替了。

ui/laf.properties文件里也包含了Spoon里很多其他属性的配置。例如, 窗口图标和Kettle home 路径的名称。

22.4.2 Kettle的二次发布 (Forking)

作为OEM版本, 实际上并没有修改Kettle的源代码。对大多数公司来说, 这种通过参数或自定义的选项来修改Kettle实际是最方便的形式。但是, 在一些场景下, 需要大范围修改Kettle的代码和基本功能。这时候, 我们就称之为一个二次发布版本 (forking)。

根据 LGPL 协议, 二次发布版本是把Kettle代码的一个拷贝作为基线开发版本, 在上面再做独立的开发, 使之成为一个独立的软件。通常大部分公司都不想维护这样的代码库, 但也有公司想这么做。

最有名的可能就是GeoKettle了。GeoKettle是Spatialytics.org 社区的一个开源项目。该项目的创建人员是Dr. Thierry Badard和Luc Vaillencourt, 他们在地理信息软件方面有丰富的开发经验。GeoKettle给Kettle增加了GIS功能 (Graphical Information Systems), 例如从Oracle Spatial、PostGIS (<http://postgis.refractory.net/>) 和带 ESRI shape 文件的MySQL中读取地理数据的能力。这个项目也增加了空间系统管理和坐标转换等功能。因为不可能通过Kettle的标准插件来实现这些功能, 开发人员就复制了Kettle代码作为基线版本, 然后再开发。GeoKettle也是LGPL 协议。通常这个版本会随着 Kettle每个版本的发布做一些调整, 一般在几个星期以后, 也发布一个版本, 已使它和Kettle兼容。因为使用空间数据的数据仓库非常有限, 所以这个二次发布可以认为是一个友好的, 不具有竞争性的二次发布。

过去也有一些个人或公司修改了代码, 然后把 Kettle二次发布。有时候, 代码也随着二进制程序一起发布; 但有时候, 并没有发布代码。很多公司在小范围内使用自己维护的二次发布的Kettle版本。根据 LGPL 协议, 使用这些代码没有问题。但要知道, 一个二次发布的Kettle版本, 其他人有权利索要代码。

同时也要知道, 想成功维护一个 Kettle二次发布的版本, 要投入大量的人力, 包括开发和翻译团队。每年都有大大小小的数千次的变化, 所以要维护好一个二次发布版本并不是很容易的事情。这也就是为什么很多二次发布版本最后慢慢变得陈旧或消失的原因。我们怀疑, 目前仍旧存在的少数 Kettle二次发布的版本, 只是在Kettle上面增加了一些插件, 换了一个名称。因为Kettle二次发布版本没有支持, 所以也不能确定是否打了哪个补丁。

22.5 小结

本章介绍了在应用程序中集成Kettle的不同方式，同时也举了几个例子。下面是本章学习的主要内容：

- 什么是LGPL协议，使用 Kettle API开发是否符合LGPL协议要求。
- 如何获得和编译Kettle的代码。
- 如何使用Java API 执行转换和作业，包括一些更高级的功能：如何从步骤里抽取数据、如何把数据注入到一个转换里。
- 如何创建你自己的Kettle OEM版本。
- 关于 GeoKettle和其他的Kettle二次发布版本。

22.1.3 简介

本节我们介绍开发Kettle插件的一些概念。

第23章 扩展Kettle

本书最后一章将介绍如何扩展Kettle的功能, 如何开发Kettle插件。34个ETL子系统中并没有这部分内容, 因为这个主题并不属于ETL子系统。但是, 你开发的任何插件都会属于本书介绍的ETL子系统, 无论是抽取数据的插件还是生成文档的插件。

尽管Kettle包括了很多步骤和作业项, 但有时我们还需要扩展Kettle, 通常是让Kettle能支持第三方的技术或新出现的技术。

我们先看Kettle的插件架构、不同类型的插件、Kettle如何加载插件。然后, 学习如何构建自己的工程来开发新的步骤、作业项或分区方法, 等等。我们详细介绍不同类型插件用到的类和方法。

23.1 插件架构

我们先看Kettle的插件架构。你将了解到有哪些类型插件, 如何在运行时加载插件。

第22章介绍过, 所有的开源软件, 包括Kettle, 都可以扩展。因为Kettle是LGPL协议, 每个人都可以获取Kettle代码, 修改和发布一个新的版本, 在第22章, 我们称之为“二次发布”。

除了扩展Kettle代码本身, Kettle的插件架构可以不用修改Kettle本身的代码, 通过一些独立的代码就可以扩展Kettle的功能。我们把这些独立的代码称为插件。Kettle可以动态加载并运行这些插件。

23.1.1 插件类型

Kettle有下面几种插件类型（下面的插件是Kettle 4.0的插件类型，在Kettle 4.4版本中，还包含了视图插件和大数据插件。——译者注）。

- 转换步骤插件：在Kettle转换中使用的步骤，用来处理数据行。
- 作业项插件：在Kettle作业中使用的作业项，用来实现某个任务。
- 分区方法插件：利用输入字段的值指定自己的分区规则
- 数据库类型插件：用来扩展不同的数据库类型。
- 资源库类型插件：可以把Kettle元数据保存为自定义类型或格式。

说明：除了这些类型，还有Spoon 类型的插件，可以把功能扩展到Spoon，本书不介绍这个功能。

23.1.2 架构

从功能上看，Kettle内部的对象和外部插件没有任何区别。因为它们使用的API 都是一样的，它们只是在运行时的加载方式不同。

从Kettle 4以后，Kettle内部有一个插件注册系统，它负责加载各种内部和外部插件。插件有以下两个标识属性。

- 插件类型：由PluginTypeInterface 接口定义。例如StepPluginType、JobEntryPluginType、PartitionerPluginType和RepositoryPluginType。
- 插件ID：这是一个字符串数组，用来唯一标识一个插件。因为旧的插件可以被新的插件代替，一个插件可以有多个ID。在大多数情况下，插件只使用一个单一的字符串，如TableInput是“表输入”步骤的ID，MYSQL是MySQL数据库类型的ID。

当 Kettle环境初始化以后，插件注册系统首先加载所有的内部对象，Kettle读取下面的配置文件来加载内部对象，这些配置文件位于Kettle的.jar文件中。

- kettle-steps.xml：内部转换步骤。
- kettle-job-entries.xml：内部作业项。
- kettle-partition-plugins.xml：内部分区类型。
- kettle-database-types.xml：内部数据库类型。
- kettle-repositories.xml：内部资源库类型。

插件注册系统加载了所有的内部对象后，就要搜索可用的外部插件。通过浏览plugins/目录的各个子目录下的.jar 文件来完成。它搜索特定的Kettle annotations来判断一个类是否是插件。加载过程将在本章的后面介绍。

因为在内部对象加载后才加载插件，所以插件会替代相同 ID的已加载的内部对象。例如，你创建了插件，插件的ID是TableInput，就可以替换Kettle标准的“表输入”步骤。这个功能可以让你用插件替换Kettle内置的步骤。可以通过子类继承的方式，直接扩展已有步骤的某些功能。

23.1.3 前提

本节我们介绍开发Kettle插件的一些前提条件。

Kettle API文档

每个插件都实现了某种插件的接口（API）。这些API包括一组Java接口和接口的一些基本实现（抽象类），用户可以直接继承这些抽象类。插件通过接口里的方法和Kettle的主程序交换数据。

插件API的接口和类都使用 javadoc 生成了文档。开发插件时可以参考这些文档。这些文档可以从下面的地址下载：<http://javadoc.pentaho.com/kettle>。

库

除了一些基本的库，开发插件还需要Eclipse SWT 库（Standard Widget Toolkit），用来开发用户界面，这些库包括：

- commands.jar
- common.jar
- jface.jar
- runtime.jar
- swt.jar

在Kettle的libswt/目录下可以找到这些jar文件。注意这里有一个swt.jar 文件，每个不同的操作系统都需要不同的swt.jar文件；在libswt/目录下每个操作系统都有对应的目录（如linux、osx和win32）。在开发时，需要使用和你的操作系统对应的swt.jar文件。

集成开发环境

集成开发环境（IDE）可以极大简化开发任务。它帮助你识别项目源代码，导航相关源代码，提供生成代码向导，运行，调试等。有很多种IDE，我们使用最流行的Eclipse。Eclipse是开源的IDE，适合Java 开发环境。

从<http://www.eclipse.org/downloads/>可以下载Eclipse IDE。这里有很多版本，我们建议使用带Eclipse IDE for Java Developer标签的IDE。

安装Eclipse 项目

在Eclipse 里创建完一个项目后，在源代码目录下（src/）创建一个包，包名为org.kettlesolutions.plugin，如图23-1所示。

在创建的包下面根据插件的类型和名字创建子包，如org.kettlesolutions.plugin.step.helloworld。

然后在工程根目录下创建 libext/和libswt/目录。把Kettle编译后的位于lib/下的几个jar包复制到插件的libext/目录下。把kettle的libswt/目录下的jar包复制到插件的libswt/目录下。注意只要你自己操作系统对应的swt.jar 即可。然后在Eclipse里配置项目的build路径，包含libext/和libswt/目录下的所有jar包。配置过程如图23-2所示。

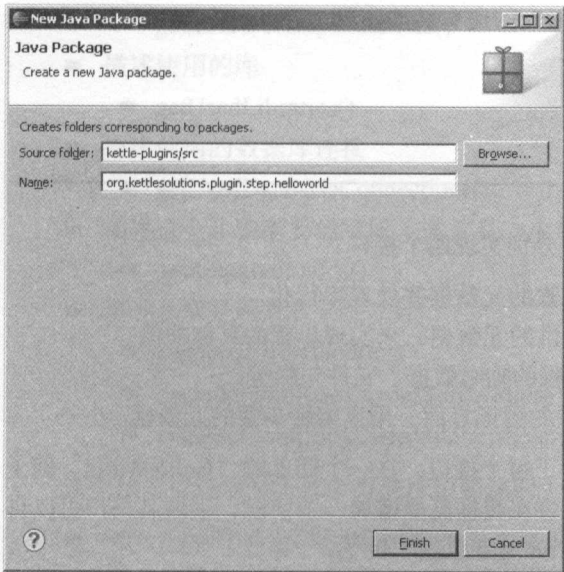


图23-1 创建一个新的Java 包

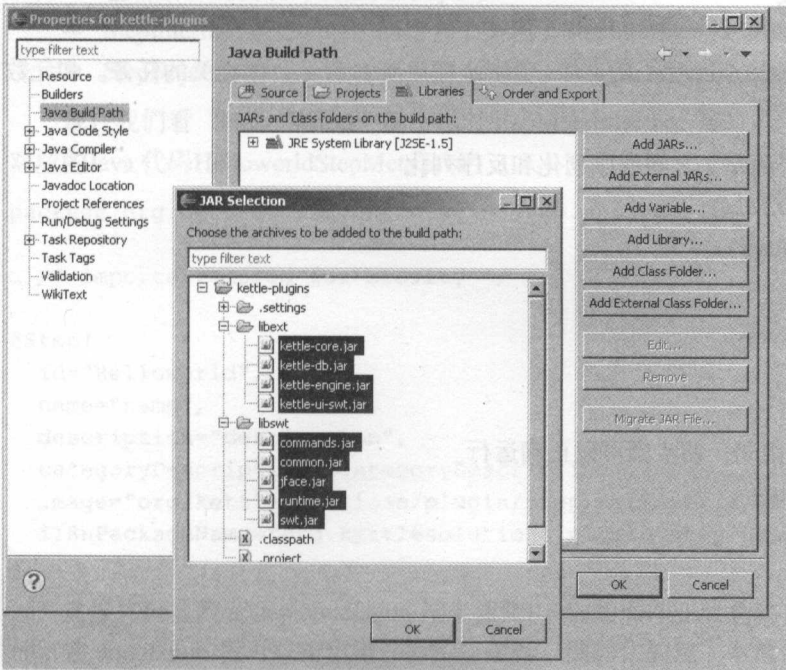


图23-2 选择依赖的.jar 文件

完成后，你就可以开发插件了。在Eclipse里查询类和接口很方便，使用 **Ctrl+Shift+T** 快捷键就可以查询（或者使用菜单 **Navigate→Open Type**）。

例子

所有Kettle内置的步骤、作业项、分区、资源库、数据库类型都是我们开发插件的例子，因为 Kettle内置的这些对象的结构和插件的结构完全一样，只是加载方式不同。Kettle的内置对象从 Kettle的.jar 文件中加载，而插件从plugins/目录下加载。

在Kettle的org.pentaho.di.trans.steps 包下是Kettle的所有转换步骤。在org.pentaho.di.job.entries 包下是Kettle的所有作业项。

23.2 转换步骤插件

转换步骤插件包括了四个Java 类，这四个类分别实现四个接口。

- StepMetaInterface: 这个接口对外提供步骤的元数据并处理串行化。
- StepInterface: 这个接口根据上面接口提供的元数据，来实现步骤的具体功能。
- StepDataInterface: 这个接口用来存储步骤的临时数据、文件句柄等。
- StepDialogInterface: 这个接口是Spoon里的图形界面，用来编辑步骤的元数据。

在本节，我们介绍这些接口的基本内容。对于每个接口，在一个简单的“Hello World”例子中提供这些类的相应实现。“Hello World”例子将在数据流里增加一个字段，字段名用户可以自定义，字段值是“Hello, world!”。最后我们介绍如何部署这个例子。

23.2.1 StepMetaInterface

接口org.pentaho.di.trans.step.StepMetaInterface负责步骤里所有和元数据相关的任务。和元数据相关的工作包括：

- 元数据和XML（或资源库）之间的序列化和反序列化
 - getXML()和loadXML()
 - saveRep()和readRep()
- 描述输出字段
 - getFields()
- 检验元数据是否正确
 - check()
- 获取步骤相应的SQL语句，使步骤可以正确运行
 - getSQLStatements()
- 给元数据设置默认值
 - setDefault()
- 完成对数据库的影响分析
 - analyseImpact()
- 描述各类输入和输出流
 - getStepIOMeta()
 - searchInfoAndTargetSteps()
 - handleStreamSelection()
 - getOptionalStreams()
 - resetStepIOMeta()
- 导出元数据资源
 - exportResources()

- getResourceDependencies()
- 描述使用的库
 - getUsedLibraries()
- 描述使用的数据库连接
 - getUsedDatabaseConnections()
- 描述这个步骤需要的字段（通常是一个数据库表）
 - getRequiredFields()
- 描述步骤是否具有某些功能
 - supportsErrorHandling()
 - excludeFromRowLayoutVerification()
 - excludeFromCopyDistributeVerification()

这个接口里还定义了几个方法来说明这四个接口如何结合到一起。

- String getDialogClassName(): 用来描述实现了StepDialogInterface接口的对话框类的名字。如果这个方法返回了null，调用类会根据实现了StepMetaInterface接口的类的类名和包名来自动生成对话框类的名字。
- StepInterface getStep(): 创建一个实现了StepInterface 接口的类。
- StepDataInterface getStepData(): 创建一个实现了StepDataInterface 接口的类。

现在我们看“Hello World”例子里对StepMetaInterface 接口的实现（在本书的下载文件里有对应的Java 代码HelloworldStepMeta.java）：

```
package org.kettlesolutions.plugin.step.helloworld;
```

```
... /* imports removed for brevity */
```

```
@Step(
    id="Helloworld",
    name="name",
    description="description",
    categoryDescription="categoryDescription",
    image="org/kettlesolutions/plugin/step/helloworld/HelloWorld.png",
    i18nPackageName="org.kettlesolutions.plugin.step.helloworld"
)
```

这段代码里的@Step annotation 用来通知Kettle的插件注册系统：这个类是一个步骤类型的插件。在 annotation 里可以指定插件的ID、图标、国际化的包、本地化的名称、类别、描述。其中后三项是资源文件里的Key，需要在资源文件里设置真正的值。i18nPackageName指定了资源文件的包名，例如我们这个例子的资源文件位于 org/kettlesolutions/plugin/step/helloworld/messages 目录下，en_US（英语，美国）的本地化资源文件是messages_en_US.properties。我们例子里的这个资源文件的内容是：

```
name=Hello world
description=A very simple step that adds "Hello world" to the incoming
    stream
categoryDescription=Plugin samples
```

注意，如果你指定了不存在的分类，Spoon 会创建这个分类，并在Spoon的分类树的最上方

显示这个分类。

最后, annotation里的image标签指定了插件的图标。需要32*32 像素的PNG 文件, 可以使用透明样式。

后面的代码行说明这个类实现了StepMetaInterface 接口。在BaseStepMeta抽象类里定义了这个接口的很多默认实现, 可以直接继承这个抽象类, 然后把工作集中在插件特有的功能上。

```
public class HelloworldStepMeta extends BaseStepMeta
    implements StepMetaInterface {
```

下面定义的PKG 变量说明了messages包的位置, 在messages包里有各种国际化的资源文件。在本章后面经常要看到的BaseMessages.getString()方法, 就是根据软件的国际化设置, 从不同的资源文件中获取文字。PKG 变量通常位于类的最上方, 被国际化图形工具使用, 通过国际化图形工具, 国际化人员可以编辑不同的国际化资源文件。所以我们会在很多Kettle代码里看见这样的结构:

```
private static Class<?> PKG = HelloworldStep.class; //for i18n
```

下面的fieldname变量用来保存用户输入的字段名: 保存 “ “Hello, world!” 字符串的字段名。

```
/**
 * @return the fieldName
 */
public String getFieldName() {
    return fieldName;
}

/**
 * @param fieldName the fieldName to set
 */
public void setFieldName(String fieldName) {
    this.fieldName = fieldName;
}
```

下面的代码用于验证用户是否在对话框里输入了字段名, 并把验证结果添加到检验转换时出现的问题列表里。(最好要检验用户输入的所有选项, 而不只是容易出错的选项。)

```
/**
 * checks parameters, adds result to List<CheckResultInterface>
 * used in Action > Verify transformation
 */
public void check(List<CheckResultInterface> remarks,
    TransMeta transMeta, StepMeta stepMeta,
    RowMetaInterface prev, String input[], String output[],
    RowMetaInterface info) {

    if (Const.isEmpty(fieldName)) {
        CheckResultInterface error = new CheckResult(
            CheckResult.TYPE_RESULT_ERROR,
            BaseMessages.getString(PKG, "HelloworldMeta.CHECK_ERR_NO_FIELD"),
            stepMeta
        );
        remarks.add(error);
```



```

    } else {
        CheckResultInterface ok = new CheckResult(
            CheckResult.TYPE_RESULT_OK,
            BaseMessages.getString(PKG, "HelloworldMeta.CHECK_OK_FIELD"),
            stepMeta
        );
        remarks.add(ok);
    }
}

```

前面介绍过，`getStep()`、`getStepData()`和`getDialogClassName()`方法提供了与这个步骤里其他三个接口之间的桥梁。

```

/**
 * creates a new instance of the step (factory)
 */
public StepInterface getStep(StepMeta stepMeta,
    StepDataInterface stepDataInterface,
    int copyNr, TransMeta transMeta, Trans trans) {
    return new HelloworldStep(stepMeta, stepDataInterface,
        copyNr, transMeta, trans);
}

/**
 * creates new instance of the step data (factory)
 */
public StepDataInterface getStepData() {
    return new HelloworldStepData();
}

public String getDialogClassName() {
    return HelloworldStepDialog.class.getName();
}

```

下面的四个方法`loadXML()`、`getXML()`、`readRep()`和`saveRep()`把元数据保存到XML文件或资源库里，或者从XML文件或资源库读取元数据。保存到文件的方法利用了像 XStream (<http://xstream.codehaus.org/>) 这样的XML 串行化技术。

```

public enum Tag {
    field_name,
};

/**
 * deserialize from xml
 * databases = list of available connections
 * counters = list of sequence steps
 */
public void loadXML(Node stepDomNode, List<DatabaseMeta> databases,
    Map<String, Counter> sequenceCounters
    ) throws KettleXMLException {
    fieldName = XMLHandler.getTagValue(stepDomNode, Tag.field_name.name());
}

```

```

public String getXML() throws KettleException {
    StringBuilder xml = new StringBuilder();
    xml.append(XMLHandler.addTagValue(Tag.field_name.name(), fieldName));
    return xml.toString();
}

/**
 * De-serialize from repository (see loadXML)
 */
public void readRep(Repository repository, ObjectId stepIdInRepository,
    List<DatabaseMeta> databases,
    Map<String, Counter> sequenceCounters
    ) throws KettleException {

    fieldName = repository.getStepAttributeString(
        stepIdInRepository,
        Tag.field_name.name()
    );
}

/**
 * serialize to repository
 */
public void saveRep(Repository repository, ObjectId idOfTransformation,
    ObjectId idOfStep) throws KettleException {
    repository.saveStepAttribute(idOfTransformation, idOfStep,
        Tag.field_name.name(), fieldName);
}

```

下面的setDefault()方法给变量设置默认值:

```

/**
 * initialize parameters to default
 */
public void setDefault() {
    fieldName = "helloField";
}

```

下面的getFields()方法非常重要,因为它描述了输出数据行的结构。这个方法需要修改inputRowMeta对象,使这个对象和输出格式匹配。Spoon 和后面的步骤都需要知道这个步骤要输出哪些字段。最常见的一种方法,可以给输出的RowMetaInterface对象添加一个ValueMetaInterface对象。在ValueMetaInterface对象里设置的信息越详细越好,可以设置的信息包括数据类型、长度、精度、格式掩码,等等。添加的字段描述元信息越多,后面生成的SQL就越准确。

```

public void getFields( RowMetaInterface inputRowMeta, String name,
    RowMetaInterface[] info, StepMeta nextStep,
    VariableSpace space) throws KettleStepException {
    String realFieldName = fieldName;
    ValueMetaInterface field =

```

```
new ValueMeta(realFieldName, ValueMetaInterface.TYPE_STRING);
field.setOrigin(name);
inputRowMeta.addValueMeta(field);
}
```

值的元数据 (Value Metadata)

值的元数据使用ValueMetaInterface接口描述数据流里的一个字段。这个接口里定义了字段的名字、数据类型、长度、精度，等等。下面的例子用于创建一个ValueMetaInterface对象。

```
ValueMetaInterface dateMeta = new ValueMeta(
    "birthdate",
    ValueMetaInterface.TYPE_DATE
);
dateMeta.setConversionMask("yyyy/MM/dd");
```

这个接口也负责转换数据格式。我们建议使用ValueMetaInterface接口来完成所有数据转换的工作。例如，日期类型的数据，如果想把它转换为 dateMeta对象里定义的字符串格式，可以用下面的代码：

```
// java.util.Date birthDate

String birthDateString = dateMeta.getString(birthDate);
```

ValueMeta类负责转换。因为有ValueMetaInterface进行数据类型的转换，所以你不用再去额外的数据类型转换的工作。

使用ValueMetaInterface接口时还要注意数据对象是否为 null。从上一个步骤可以接收到一个数据对象和一个描述数据对象的ValueMetaInterface对象。我们要检查这个数据对象是否为null，在某些情况下如果数据对象为空是不正确的。例如：

- 数据对象是String类型，有10个空格，Value Metadata 需要trim 这个字符串。
- 在Value Metadata里已经定义了从文本文件里加载的数据，要延迟转换为字符串。所以数据要由二进制的格式（原始数据格式），转换为字符串格式，然后再转换为其他格式的数据。关于延迟加载和转换的更多内容请参考第15章。

一般使用下面的方法检查数据对象是否为空：

```
boolean n = valueMeta.isNull(valueData);
```

重要：要保证传给ValueMetaInterface对象的数据是在元数据里定义的数据类型。表23-1说明了ValueMetaInterface里定义的数据类型和Java数据类型的对应关系。

表23-1 Kettle元数据类型和Java 里数据类型的对应关系

| Value Meta Type | Java Class |
|---------------------------------|-------------------|
| ValueMetaInterface. TYPE_STRING | java.lang.String |
| ValueMetaInterface.TYPE_DATE | java.util.Date |
| ValueMetaInterface.TYPE_BOOLEAN | java.lang.Boolean |
| ValueMetaInterface.TYPE_NUMBER | java.lang.Double |
| ValueMetaInterface.TYPE_INTEGER | java.lang.Long |

续表

| Value Meta Type | Java Class |
|-----------------------------------|----------------------|
| ValueMetaInterface.TYPE_BIGNUMBER | java.math.BigDecimal |
| ValueMetaInterface.TYPE_BINARY | byte[] |

行的元数据 (Row Metadata)

行的元数据使用RowMetaInterface接口来描述数据行的元数据，而不是一个列的元数据。实际上，RowMetaInterface的类里包含了一组ValueMetaInterface。另外还包括了一些方法来操作行元数据，例如查询值、检查值是否存在、替换值的元数据等。

行的元数据里唯一的规则就是一行里列的名字必须唯一。当你添加了一个新列时，如果新列的名字和已有列的名字相同，列名后面会自动加上 “_2” 后缀。如果再加一个同名的列，会自动加上 “_3” 后缀，等等。

因为在步骤里通常是和数据行打交道，所以从数据行里直接取数据会更方便。可以使用很多类似于getNumber()、getString() 这样的方法直接从数据行取数据。例如，销售数据存储在第四列里，可以用下面的代码获取这个数据：

```
Double sales = getInputRowMeta().getNumber(rowData, 3);
```

通过索引获取数据是最快的方式。通过 indexOfValue()方法可以获取列在一行里的索引。这个方法扫描列数组，速度并不快。所以，如果要处理所有数据行，我们建议只查询一次列索引。一般是在步骤接收到第一行数据时，就查询列索引，将查询到的列索引保存起来，供后面的数据行使用。

23.2.2 StepDataInterface

实现了org.pentaho.di.trans.step.StepDataInterface接口的类用来维护步骤的执行状态，以及存储临时对象。例如，可以把输出行的元数据、数据库连接、输入输出流等存储到这个对象里。

23.2.3 StepDialogInterface

实现了org.pentaho.di.trans.step.StepDialogInterface接口的类用来提供一个用户界面，用户通过这个界面输入元数据（转换参数）。用户界面就是一个对话框。这个接口里只包含了类似open()和setRepository()等的几个简单的方法。

Eclipse SWT

Kettle使用Eclipse SWT作为界面开发包，所以你也要使用SWT来开发对话框窗口。SWT为不同的操作系统Windows、OS X、Linux和UNIX提供了一个抽象层。所以SWT的图形界面和操作系统其他的程序的界面风格非常相近。

在开始进行SWT开发之前，建议先访问SWT主页以了解更多内容<http://www.eclipse.org/swt>。在 SWT的网站上，你可以了解到SWT能做出什么样的界面效果：

- SWT控件页，<http://www.eclipse.org/swt/widgets/>，给出了你能使用的所有控件。

■ SWT 样例页, <http://www.eclipse.org/swt/snippets/>, 给出了很多Java 代码例子。
最好的资源就是Kettle里150个内置步骤的对话框源代码。

窗体布局

如果你看过步骤对话框的源代码, 你就会发现窗体类里有很多烦琐的代码。这些代码确保Kettle可以在各种不同操作系统下以最合适的方式展现窗体。可以发现窗体里的大部分代码都和布局以及控件位置有关。

FormLayout是SWT 里经常看到的布局方式。程序员可以通过FormLayout指定控件的百分比、偏移。下面是我们例子里的窗口布局的代码 (HelloworldStepDialog.java) :

```
Label label =new Label(shell, SWT.RIGHT);
label.setText("Hello");
props.setLook(wlStepname);
FormData fdLabel=new FormData();
fdLabel.left = new FormAttachment(0, 0);
fdLabel.right= new FormAttachment(50, -10);
fdLabel.top = new FormAttachment(0, 25);
label.setLayoutData(fdLabel);
```

这些是我们经常看到的代码, 所以我们详细解释一下。第一行创建了一个新的标签控件, 控件里文本靠右对齐:

```
Label label =new Label(shell, SWT.RIGHT);
```

下面把字符串“Hello”放到标签里:

```
label.setText("Hello");
```

下面一行为控件设置用户定义的背景色和字体:

```
props.setLook(wlStepname);
```

下面几行将标签的左侧和对话框的最左侧对齐, 把标签的右侧放在对话框中间 (50%) 的左侧10个像素的位置。标签的顶部放在距离对话框顶部25个像素的位置。

```
FormData fdLabel=new FormData();
fdLabel.left = new FormAttachment(0, 0);
fdLabel.right= new FormAttachment(50, -10);
fdLabel.top = new FormAttachment(0, 25);
label.setLayoutData(fdLabel);
```

FormAttachment的第一个参数除了是百分比之外, 还可以是另一个控件。通过这种方式可以指定一个插件相对另一个插件的位置。

如果不指定控件的底部位置, SWT将使用控件的高度来定位, 控件字体不同, 高度不同。如果你想把一个文本输入控件放在标签控件的下面10个像素的位置, 可以这么写:

```
fdLabel2.top = new FormAttachment(label, 10);
```

简而言之, 不要感到痛苦; 图形用户界面的代码都比较烦琐, 但代码并不复杂。

Kettle UI元素

除了标准的SWT组件，还可以使用Kettle自带的一些控件，Kettle开发人员的工作可以更简单一些。Kettle自带的组件包括以下一些。

- TableView: 这是一个数据表格组件，支持排序、选择、键盘快捷键和撤销/重做，以及右键菜单。
- TextVar: 这是一个支持变量的文本输入框，这个输入框的右上角有一个\$符号。用户可以通过“Ctrl+Alt+空格”的方式，在弹出的下拉列表中选择变量。其他功能和普通的文本输入框相同。
- ComboVar: 标准的组合下拉列表，支持变量。
- ConditionEditor: 过滤行步骤里使用的输入条件控件。

另外还有很多常用的对话框帮你完成相应的工作，如下所示。

- EnterListDialog: 从字符串列表里选择一个或多个字符串。左侧显示字符串列表，右侧是选中的字符串，并提供把字符串从左侧移动到右侧的按钮。
- EnterNumberDialog: 用户可以输入数字。
- EnterPasswordDialog: 让用户输入密码。
- EnterSelectionDialog: 通过高亮显示，从列表里选择多项。
- EnterMappingDialog: 输入两组字符串的映射。
- PreviewRowsDialog: 在对话框里预览一组数据行。
- SQLEditor: 一个简单的SQL编辑器，可以输入查询和DDL。
- ErrorDialog: 显示异常信息，列出详细的错误栈对话框。

Hello World例子对话框

现在我们已经基本了解了SWT以及对话框的布局方式，再看看我们的例子，下面的代码是HelloWorldStepDialog.java里的例子。

代码的第一部分是初始化元数据对象以及步骤对话框类的父类：

```
public class HelloworldStepDialog extends BaseStepDialog implements
    StepDialogInterface {
```

```
    private static Class<?> PKG = HelloworldStepMeta.class;
    private HelloworldStepMeta input;
```

```
    private TextVar wFieldname;
```

```
    public HelloworldStepDialog(Shell parent, Object baseStepMeta,
        TransMeta transMeta, String stepname) {
        super(parent, (BaseStepMeta)baseStepMeta, transMeta, stepname);
        input = (HelloworldStepMeta)baseStepMeta;
    }
```

在下面的open()方法里创建对话框里的所有控件。SWT使用事件监听模式，可以为控件创建各种监听方法，以响应控件内容的变化和用户的动作。

```
    public String open() {
```

```

Shell parent = getParent();
Display display = parent.getDisplay();

shell = new Shell(parent, SWT.DIALOG_TRIM | SWT.RESIZE | SWT.MIN |
    SWT.MAX);
props.setLook(shell);
setShellImage(shell, input);

ModifyListener lsMod = new ModifyListener()
{
    public void modifyText(ModifyEvent e)
    {
        input.setChanged();
    }
};
changed = input.hasChanged();

```

下面的代码说明窗体里的控件将使用formLayout的布局方式。

```

FormLayout formLayout = new FormLayout ();
formLayout.marginWidth = Const.FORM_MARGIN;
formLayout.marginHeight = Const.FORM_MARGIN;

shell.setLayout(formLayout);

```

所有控件的右侧使用一个自定义的百分比对齐：props.getMiddlePct()。控件之间的间距使用一个常量，常量值是4像素。

```

shell.setText(
    BaseMessages.getString(PKG, "HelloworldDialog.Shell.Title"));

int middle = props.getMiddlePct();
int margin = Const.MARGIN;

```

下面的代码在对话框的最上面添加了一行步骤名称标签和输入文本框：

```

// Stepname line
wlStepname=new Label(shell, SWT.RIGHT);
wlStepname.setText(
    BaseMessages.getString(PKG, "HelloworldDialog.Stepname.Label"));
props.setLook(wlStepname);
fdlStepname=new FormData();
fdlStepname.left = new FormAttachment(0, 0);
fdlStepname.right= new FormAttachment(middle, -margin);
fdlStepname.top = new FormAttachment(0, margin);
wlStepname.setLayoutData(fdlStepname);
wStepname=new Text(shell, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
wStepname.setText(stepname);
props.setLook(wStepname);
wStepname.addModifyListener(lsMod);
fdStepname=new FormData();
fdStepname.left = new FormAttachment(middle, 0);
fdStepname.top = new FormAttachment(0, margin);
fdStepname.right= new FormAttachment(100, 0);

```

```
wStepname.setLayoutData(fdStepname);
```

```
Control lastControl = wStepname;
```

下面是新增输出列的列名设置的输入框:

```
// Fieldname line
```

```
Label wlFieldname = new Label(shell, SWT.RIGHT);
```

```
wlFieldname.setText(
```

```
    BaseMessages.getString(PKG, "HelloworldDialog.Fieldname.Label"));
```

```
props.setLook(wlFieldname);
```

```
FormData fdlFieldname = new FormData();
```

```
fdlFieldname.left = new FormAttachment(0, 0);
```

```
fdlFieldname.right= new FormAttachment(middle, -margin);
```

```
fdlFieldname.top = new FormAttachment(lastControl, margin);
```

```
wlFieldname.setLayoutData(fdlFieldname);
```

```
wFieldname =
```

```
    new TextVar(transMeta, shell, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
```

```
props.setLook(wFieldname);
```

```
wFieldname.addModifyListener(lsMod);
```

```
FormData fdFieldname = new FormData();
```

```
fdFieldname.left = new FormAttachment(middle, 0);
```

```
fdFieldname.top = new FormAttachment(lastControl, margin);
```

```
fdFieldname.right= new FormAttachment(100, 0);
```

```
wFieldname.setLayoutData(fdFieldname);
```

```
lastControl = wFieldname;
```

然后创建两个按钮,“确认”和“取消”按钮,以及按钮单击事件的监听方法,把按钮放在对话框的最下面:

```
// Some buttons
```

```
wOK=new Button(shell, SWT.PUSH);
```

```
wOK.setText(BaseMessages.getString(PKG, "System.Button.OK"));
```

```
wCancel=new Button(shell, SWT.PUSH);
```

```
wCancel.setText(
```

```
    BaseMessages.getString(PKG, "System.Button.Cancel"));
```

```
setButtonPositions(
```

```
    new Button[] { wOK, wCancel }, margin, lastControl);
```

```
// Add listeners
```

```
lsCancel = new Listener() { public void handleEvent(Event e) {  
    cancel(); } };
```

```
lsOK = new Listener() { public void handleEvent(Event e) {  
    ok(); } };
```

```
wCancel.addListener(SWT.Selection, lsCancel);
```

```
wOK.addListener (SWT.Selection, lsOK );
```

下面的代码做了两件事情,上部代码可以保证当步骤名称或输出字段名称的输入框在编辑状态时,单击“确定”按钮,正在编辑的内容不会丢失;下部的代码保证了窗口在非正常关闭时(没有使用“确定”或“取消”按钮关闭),取消用户的编辑。

```
lsDef=new SelectionAdapter() {
```

```
    public void widgetDefaultSelected(SelectionEvent e) { ok(); } };
```



```
wStepname.addSelectionListener( lsDef );
wFieldname.addSelectionListener( lsDef );

// Detect X or ALT-F4 or something that kills this window...
shell.addShellListener(new ShellAdapter() {
    public void shellClosed(ShellEvent e) {
        cancel();
    }
});
```

下面的代码把数据从步骤的元数据对象里复制到窗口的控件里：

```
// Populate the data of the controls
//
getData();
```

窗口的大小和位置将根据窗口的自然属性、上次窗口大小和位置，以及显示屏的大小自动设置。

```
// Set the shell size, based upon previous time...
setSize();

input.setChanged(changed);

shell.open();
while (!shell.isDisposed())
{
    if (!display.readAndDispatch()) display.sleep();
}
return stepname;
}
```

为了防止用户向控件里输入空值，Kettle提供了一个静态方法来检查空值，Const.NVL()：

```
/**
 * Copy information from the meta-data input to the dialog fields.
 */
public void getData()
{
    wFieldname.setText(Const.NVL(input.getFieldName(), ""));
    wStepname.selectAll();
}
```

最后，单击OK按钮后，把控件里用户输入的数据都写入到步骤的元数据对象中：

```
private void cancel()
{
    stepname=null;
    input.setChanged(changed);
    dispose();
}

private void ok()
{
    if (Const.isEmpty(wStepname.getText())) return;
```

```

stepname = wStepname.getText(); // return value
input.setFieldName(wFieldname.getText());
dispose();
}
}

```

23.2.4 StepInterface

这个类实现了org.pentaho.di.trans.step.StepInterface接口，这个类读取上个步骤传来的数据行，利用StepMetaInterface对象里定义的元数据，逐行转换和处理上个步骤传来的数据行。Kettle引擎直接使用这个接口里的很多方法来执行转换过程，但大部分方法都已经由BaseStep类实现了，通常开发人员只需要重载其中的几个方法。

- `init()`: 步骤初始化方法，用来初始化一个步骤。初始化的结果是一个true 或者false的Boolean 值。如果你的步骤没有任何初始化的工作，可以不用重载这个方法。
- `dispose()`: 如果有需要释放的资源，可以在`dispose()`方法里释放，例如可以关闭数据库连接、释放文件、清除缓存等。在转换的最后Kettle引擎会调用这个方法。如果没有需要释放或清除的资源，可以不用重载这个方法。
- `processRow()`: 这个方法，是步骤实际工作的地方。只要这个方法返回true，转换引擎就会重复调用这个方法。

下面是Hello World例子实现的StepInterface 接口（HelloworldStep.java）：

```

package org.kettlesolutions.plugin.step.helloworld;

... /* imports removed for brevity */

public class HelloworldStep extends BaseStep implements StepInterface {

```

前面介绍过，可以直接集成BaseStep 类，BaseStep 抽象类已经实现了接口里的很多方法，我们只要覆盖需要修改的方法即可。

类的构造函数通常直接把参数传递给BaseStep父类。由父类里的方法来构造对象，然后可以直接使用类似transMeta 这样的对象。

```

public HelloworldStep(StepMeta stepMeta,
    StepDataInterface stepDataInterface,
    int copyNr, TransMeta transMeta, Trans trans) {
    super(stepMeta, stepDataInterface, copyNr, transMeta, trans);
}

```

`getRow()`方法从上一个步骤获取一行数据。如果没有更多要获取的数据行，这个方法就会返回null。如果前面的步骤不能及时提供数据，这个方法就会阻塞，直到有可用的数据行。这样这个步骤的速度就会降低，也会影响到转换里其他步骤的速度。关于性能请参考第15章。

```

public boolean processRow(StepMetaInterface smi, StepDataInterface sdi
    ) throws KettleException {

    HelloworldStepMeta meta = (HelloworldStepMeta) smi;

```

```

HelloworldStepData data = (HelloworldStepData) sdi;

Object[] row = getRow();
if (row==null) {
    setOutputDone();
    return false;
}

if (first) {
    first=false;
    data.outputRowMeta = getInputRowMeta().clone();
    meta.getFields(data.outputRowMeta,
        getStepname(), null, null, this);
}

String value = "Hello, world!";

Object[] outputRow = RowDataUtil.addValueData(row,
    getInputRowMeta().size(), value);

putRow(data.outputRowMeta, outputRow);

return true;
}
}

```

从性能上考虑，getRow()方法不提供数据行的元数据，只提供上个步骤输出的数据。可以使用getInputRowMeta()方法获取元数据，元数据只获取一次即可，所以在first代码块里获取元数据。

setOutputDone()方法用来通知其他的步骤，本步骤已经没有输出数据行。下一个步骤如果再调用getRow()方法就会返回null，转换也不再调用processRow()方法。

```

Object[] row = getRow();
if (row==null) {
    setOutputDone();
    return false;
}

```

如果要把数据传到下一个步骤，要使用putRow()方法。除了输出数据，还要输出RowMetaInterface元数据。上一章介绍过如何生成输出行的元数据对象：

```

data.outputRowMeta= getInputRowMeta().clone();
meta.getFields(data.outputRowMeta,getStepname(),null, null, this);

```

第一行使用 clone()方法把输入行的元数据结构复制给输出行。输出行的元数据结构是在输入行的基础上增加一个字段，但构造输出行的元数据结构只能构造一次，因为所有输出数据行的结构都是一样的，产生了输出行以后，元数据结构就不能再变化。所以输出行的元数据结构也在first代码块里构造。first是一个内部成员，first代码块里的代码只在处理第一行数据时执行。上面代码的最后一行，给输出数据增加了一个字段。

下面的代码，就是把数据写到输出流里：

```
String value = "Hello, world!";

Object[] outputRow = RowDataUtil.addValueData(row,
    getInputRowMeta().size(), value);

putRow(data.outputRowMeta, outputRow);
```

从性能角度考虑，数据行实际就是Java数组。为了开发方便，可以使用RowDataUtil类提供的一些静态方法来操作数据。使用RowDataUtil静态方法复制数据，还可以提高性能。

从指定的步骤读取数据行

如果你想从前面的某个特定的步骤读取数据行，例如“流查询”步骤，可以使用getRowFrom()方法。

```
RowSet rowSet = findInputRowSet(Source Step Name);
Object[] rowData=getRowFrom(rowSet);
```

还可以通过rowSet对象获得数据行的元数据：

```
RowMetaInterface rowMeta = rowSet.getRowMeta();
```

把数据行写入到指定的步骤

如果想把数据写入到某个特定的步骤，例如“过滤”步骤，可以使用putRowTo()方法：

```
RowSet rowSet = findOutputRowSet(Target Step Name);
...
putRowTo(outputRowMeta, rowData, rowSet);
```

很明显，输入和输出的RowSet对象只需获得一次即可，这样才更有效率。

把数据行写入到错误处理步骤

如果想让你的步骤支持错误处理，而且元数据类返回的supportErrorHandling()方法返回了true，就可以把数据输出到错误处理步骤里。下面是使用putError()方法的例子：

```
Object[] rowData = getRow();
...
try {
    ...
    putRow(...);
} catch(Exception e) {
    if (getStepMeta().isDoingErrorHandling()) {
        putError(getInputRowMeta(), rowData, 1, e.getMessage(),
            errorMessage, errorFieldname, errorCode);
    } else {
        throw(e);
    }
}
```

从例子里可以看到，这段代码把错误的行数、错误字段名、消息、错误编码都传递给错误处理步骤。错误处理的其他工作都自动完成了。

识别一个步骤拷贝

因为一个步骤可以有多份拷贝同时执行，有时需要识别出正在使用的是哪个步骤拷贝，可以用下面几个方法。

- `getCopy()`：获得拷贝号。拷贝号可以唯一标识出步骤的一个拷贝，拷贝号的取值范围是0~N， $N = \text{getStepMeta().getCopies()} - 1$ 。
- `getUniqueStepNrAcrossSlaves()`：获得在集群模式下运行的步骤拷贝号。
- `getUniqueStepCountAcrossSlaves()`：获得在集群模式下运行的步骤拷贝总数。

通过这些方法可以把一个步骤的工作分配给多份拷贝去完成。例如“CSV 文件输入”和“固定宽度文件输入”步骤里都有并行读取文件的选项，这样可以把读取文件的工作放在多个拷贝里或集群里来完成。

结果反馈

在调用`getRow()`和`putRow()`方法时，引擎会自动计算两类度量值，读行数和写行数。这两类度量值可以在界面或日志中记录下来，以监控程序运行的状态。下面几个方法用来操作这两类度量值。

- `incrementLinesRead()`：增加从前面步骤读取到的行数。
- `incrementLinesWritten()`：增加写入到后面步骤中的行数。
- `incrementLinesInput()`：增加从文件、数据库、网络等资源读取到的行数。
- `incrementLinesOutput()`：增加写入到文件、数据库、网络等资源的行数。
- `incrementLinesUpdated()`：增加更新的行数。
- `incrementLinesSkipped()`：增加跳过的数据行的行数。
- `incrementLinesRejected()`：增加拒绝的数据行的行数。

这些度量值用来说明步骤执行的情况。可以在Spoon的转换度量面板里看到，也可以存到日志数据库表里，请看第14章。

使用`addResultFile()`方法，可以把步骤用到的文件保留下来，保存到结果文件列表里。结果文件列表可以被其他转换或作业项使用。例如，下面的“CSV文件输入”的代码：

```
ResultFile resultFile = new ResultFile(
    ResultFile.FILE_TYPE_GENERAL,
    fileObject,
    getTransMeta().getName(),
    getStepName()
);
resultFile.setComment("File was read by a Csv input step");
addResultFile(resultFile);
```

变量替换

如果输入框需要支持变量（关于变量见第2章），可以使用`environmentSubstitute()`方法获取变量。例如，若想在“Hello World”例子的字段名输入框里使用变量，就要把`StepMetaInterface`里的`getFields()`方法修改成下面的语句：

```
String realFieldName = space.environmentSubstitute(fieldName);
```

因为步骤本身是一个VariableSpace对象，所以也可以使用下面的语句做变量替换：

```
String value = environmentSubstitute(meta.getStringWithVariables());
```

Apache VFS

Kettle里所有操作文件的步骤，都使用Apache VFS 系统的方式操作文件。第2章详细描述了Apache VFS 系统，它不但可以从文件系统读取文件（如java.io.File），还可以从很多其他来源读取文件，如FTP服务器、.zip压缩文件，等等。

Apache VFS 里的FileObject对象提供了文件的抽象层，然后在 Kettle的KettleVFS类里还提供了一系列的静态方法，来更方便使用FileObject对象。例如下面的代码：

```
FileObject fileObject = KettleVFS.getFileObject(
    "zip:http://www.example.com/archive.zip!file.txt"
);
```

应该尽可能多地使用KettleVFS，因为它解决了或绕过了很多 Apache VFS 目前已知的问题。它也增强了SFTP协议。

步骤插件部署

部署之前，要把四个Java 源代码文件编译为class 文件。把编译好的class文件放到一个jar 包里。可以使用IDE 来做这些事情，也可以手工使用ant 脚本来做这些事情。

jar 文件应该放在Kettle的plugins/steps 目录下。也可以使用一个子目录，把所有依赖的jar 包放在插件jar包所在目录的/lib 目录下，不必再放Kettle的类路径中（Kettle的libext/目录）已经有了的jar 包。另外可以把多个插件放在一个jar 包里。

如果想在IDE 里调试插件，可以把插件元数据类的名字放在 KETTLE_PLUGIN_CLASSES 变量里（一个逗号分隔的列表）。关于这个主题的更多信息，请参考Pentaho Wiki: <http://wiki.pentaho.com/display/EAI/How+to+debug+a+Kettle+4+plugin>。

23.3 用户自定义 Java 类步骤

用户自定义 Java 类步骤（User Defined Java Class, UDJC）是Kettle 4 里的新步骤。Java 代码可以放在这个步骤里，在运行时编译和执行，这样可以提高转换的灵活性和效率。

如果学习过前面的插件开发内容，再写UDJC 步骤就很容易了。可以直接在UDJC 步骤里写init()、dispose()和processRow()方法。实际上，几乎你前面学过的关于开发插件的所有内容都可以在这个步骤里使用。唯一不同的是，这个步骤不使用对话框类和元数据类来接收用户输入的数据。

23.3.1 传递元数据

如果不使用参数，UDJC 里的代码将会非常不灵活和难以维护。所以我们需要在对话框下面输入参数和值。在UDJC 里，可以使用下面的语句获取参数：

```
String value = getParameter("TAG");
```

23.3.2 访问输入和字段

UDJC 步骤里有一个方便的get()方法用来访问输入和输出字段。下面的代码从一行数据里读取一个字符串的值:

```
String name = get(Fields.In, "last_name").getString(rowData);
```

同样可以使用set方法,把数据设置到输出字段中:

```
get(Fields.Out, "book_name").setValue(r, "KettleSolutions");
```

23.3.3 代码片段

UDJC对话框左侧的树里包括了一些代码片段。双击这些代码片段,可以把它们插入到你的代码里。这些代码片段都是一些简单的例子,告诉你UDJC 步骤能完成哪些功能。

23.3.4 例子

下面的代码例子实现了与前面“Hello World”插件例子同样的功能。在输出流里增加一列,把这列的值设置为“Hello, World!”。这个例子在本书的UDJC sample.ktr转换中。

```
int outputRowSize;

public boolean processRow(StepMetaInterface smi, StepDataInterface sdi
    ) throws KettleException
{
    Object[] r = getRow();
    if (r == null) {
        setOutputDone();
        return false;
    }

    if (first) {
        first = false;
        outputRowSize = data.outputRowMeta.size();
    }

    r = createOutputRow(r, outputRowSize);
    get(Fields.Out, "hello").setValue(r, "Hello, World!");

    // Send the row on to the next step.
    putRow(data.outputRowMeta, r);

    return true;
}
```

可以看到这里的代码,和我们前面写的StepInterface 类的代码非常相近。在UDJC 步骤下面的Fields 标签下只设置了一个名字为“hello”的输出字段。这种设置参数的方法和写插件一样灵活。但是和写插件相比,使用UDJC 需要写的代码更少,节省了开发时间。

23.4 作业项插件

写作业项插件和写步骤插件非常类似，主要的区别是只要实现两个接口就可以了。

- `JobEntryInterface`: 管理作业项的元数据和执行作业项。
- `JobEntryDialogInterface`: 作业项的对话框类，接收用户输入作业元数据。

步骤是并行执行的，作业是串行执行的，所以作业项里没有把元数据部分代码和执行部分代码分成两个类。这样开发工作也简单一些。

23.4.1 `JobEntryInterface`

`org.pentaho.di.job.entry.JobEntryInterface`接口里有下面这个最重要的方法。

- `execute()`: 这个方法执行作业项并返回“结果”对象（关于“结果”对象请参考第22章）。

其他的方法和前面介绍过的步骤插件的方法类似，这些方法包括：

- `getXML()`和`loadXML()`用来串行化到XML。
- `saveRep()`和`readRep()`用来串行化到资源库。
- `getSQLStatements()`用来生成SQL。
- `getUsedDatabaseConnections()` 用来看作业项使用了哪个数据库连接。
- `check()` 用来检查元数据。
- `getResourceDependencies()` 用来列出依赖的资源。
- `exportResources()` 导出这个作业依赖的所有资源（例如作业和转换）。
- `getDialogClassName()` 获得用来编辑元数据的对话框。

还有两个偶尔使用到的方法如下所示。

- `boolean evaluates()`: 判断作业项是否要做检验。除了极少数特殊的作业项（如“开始”作业项），这个方法都应该返回`true`。
- `boolean isUnconditional()`: 说明执行完这个作业项后，是否允许后续作业项无条件连接到这个作业项，除了极少数特殊的作业项，这个方法都应该返回`true`。

让我们看一个“Hello World”作业项的例子。这个作业项允许用户指定一个Boolean 值 `true` 或`false`。如果指定了`true`，那么作业沿着成功路线执行；如果指定了`false`，作业沿着失败的路线执行。这个作业项模拟了作业的成功和失败结果。

`@JobEntry` annotation的工作原理和`@Step` annotation的工作原理相同。插件注册系统通过annotation识别出这是一个作业项插件（参考下面的`HelloWorldJobEntry.java`代码）：

```
package org.kettlesolutions.plugin.jobentry.helloworld;
```

```
... /* imports removed for brevity */
```

```
@JobEntry(
    id="Helloworld",
    name="HelloworldJobEntry.name",
    description="HelloworldJobEntry.description",
```



```
categoryDescription="HelloworldJobEntry.category",
packageName="org.kettlesolutions.plugin.jobentry.helloworld",
image="org/kettlesolutions/plugin/step/helloworld/HelloWorld.png"
)
```

为了方便，通常直接继承JobEntryBase类，只需实现必要的方法：

```
public class HelloworldJobEntry extends JobEntryBase
implements JobEntryInterface {
```

和步骤插件类似，这个类包含了需要的元数据（参数），和处理元数据的get和set方法：

```
private boolean success;

/**
 * @return the success
 */
public boolean isSuccess() {
    return success;
}

/**
 * @param success : the success flag to set
 */
public void setSuccess(boolean success) {
    this.success = success;
}
```

执行结果是一个Result对象，这个Result对象会传递到下一个作业项。

```
public Result execute(Result prevResult, int nr)
throws KettleException {
    prevResult.setResult(success);
    return prevResult;
}
```

关于execute()方法，要记住的是一定要使用同一个Result对象，修改前面传过来的Result对象，而不能创建新的Result对象。

```
public enum Tag {
    success,
};

public void loadXML(Node entrynode, List<DatabaseMeta> databases,
    List<SlaveServer> slaveServers, Repository rep
) throws KettleXMLException {
    super.loadXML(entrynode, databases, slaveServers);
    success = "Y".equalsIgnoreCase(
        XMLHandler.getTagValue(entrynode, Tag.success.name()));
}
```

```
public String getXML() {
    StringBuilder xml = new StringBuilder();
    xml.append(super.getXML());
    xml.append(XMLHandler.addTagValue(Tag.success.name(), success));
}
```

```

23  return xml.toString();
    }

    public void loadRep(Repository rep, ObjectId idJobentry,
        List<DatabaseMeta> databases, List<SlaveServer> slaveServers
    ) throws KettleException {

        success =
            rep.getJobEntryAttributeBoolean(idJobentry, Tag.success.name());
    }

    public void saveRep(Repository rep, ObjectId idJob)
        throws KettleException {
        rep.saveJobEntryAttribute(idJob, getObjectId(),
            Tag.success.name(), success);
    }
}

```

23.4.2 JobEntryDialogInterface

作业项对话框和步骤对话框（StepDialogInterface）基本类似，我们前面已经详细描述过步骤对话框。这里唯一不同的是，你要实现 org.pentaho.di.job.entry.JobEntryDialogInterface 接口。

和步骤对话框接口不同的是，在作业项对话框接口里需要编辑 JobEntryInterface 类，而不是 StepMetaInterface类。可以直接继承 JobEntryDialog 类。如何构造和使用作业项对话框，可以参考Kettle的源代码。

部署和调试作业项插件和调试步骤插件一样，需要把 .jar文件放在 plugins/jobentries 目录下。

23.5 分区插件

第16章曾经介绍过，一行数据属于哪个分区是由分区方法决定的。通过分区插件，可以根据数据开发新的分区规则。

要实现分区插件，需要实现两个接口Partitioner 和StepDialogInterface（用后面这个接口有一定历史原因，因为对话框接口通常都包括open()方法）。

这个对话框和前面介绍的对话框的区别是：这个对话框接收Partitioner对象而不接收步骤或作业项的元数据。这个对话框的其他方面和前面介绍过的对话框都相同。所以，我们这里就不再详细介绍这个对话框了。关于这个对话框的例子请参考本章例子里的HourPartitionerDialog.java 和Kettle自带的ModPartitionerDialog。

Partitioner接口

这个接口里除了有串行化元数据到XML和资源库的那四个方法，以及getDialogClassName()方法外，还有一个重要方法要实现：getPartition()。下面的代码里，根据文件名的后两个字符来分区，后两位数据就是文本里的数据被采集到的时间。例如，文件名data-17.txt，就是下午5点到

6点之间的数据（24小时制）。对于这个文件，getPartition()方法应该返回分区号17，因为分区号从0开始，所以这是第18个分区。

在这个例子里，分区方法的元数据包含了文件名字段。这个方法还初始化分区数变量nrPartitions。这个方法继承自BasePartioner类。可以从HourPartitioner.java例子里看到下面的代码：

```
public int getPartition(RowMetaInterface rowMeta, Object[] row
    ) throws KettleException {
    init(rowMeta);
```

在我们的例子里，要有一个包含文件名的字段，若没有找到这个字段，程序会抛出异常：

```
if (partitionColumnIndex < 0) {
    partitionColumnIndex = rowMeta.indexOfValue(fieldName);
    if (partitionColumnIndex < 0) {
        throw new KettleStepException(
            BaseMessages.getString(PKG,
                "HourPartitioner.Exception.PartitioningFieldNotFound",
                fieldName,
                rowMeta.toString()));
    }
}
```

下面取得文件名，同时检查输入是否是String类型。如果不是String类型，终止转换抛出异常：

```
ValueMetaInterface valueMeta =
    rowMeta.getValueMeta(partitionColumnIndex);
Object valueData = row[partitionColumnIndex];

if (!valueMeta.isString()) {
    throw new KettleException(BaseMessages.getString(PKG,
        "HourPartitioner.Exception.NotAFilename",
        valueMeta.getName()));
}
```

下面的代码行用来计算分区号。要返回一个0到分区数减1之间的数字。为了确保这个取值范围，我们使用模运算（除法取余数）：

```
String filename = valueMeta.getString(valueData);
String hourString =
    filename.substring(filename.length()-6, filename.length()-4);
int value = Integer.parseInt(hourString);
int targetLocation = (int) (value % nrPartitions);

return targetLocation;
}
```

部署分区插件也和部署其他类型插件一样，由于历史原因，需要把.jar文件部署到和步骤一样的目录下：plugins/steps。

23.6 资源库插件

最近十年,越来越多的低价和开源的配置管理软件 (Software Configuration Management, SCM) 和内容管理软件 (Content Management Systems, CMS) 出现了。例如,流行的开源SCM软件包括CVS、Subversion和Git。在CMS领域,也有JCR (Java Content Repository)、CMIS (Content Management Interoperability Services) 以及最新的Google Wave 协议。所有这些系统都适合于Kettle元数据的存储和提取,所以 Kettle开发团队认为需要有一个抽象层,让用户实现自己的资源库类型,把元数据存储在任何SCM 或 CMS 系统中。

Repository是资源库插件要实现的接口。它覆盖的功能很广,从转换的串行化到安全访问规范。因为这类插件涉及的功能太广泛,所以这个插件超出了大部分ETL 开发人员的关心范围。但是,如果你对写一个Kettle资源库感兴趣,也可以看看这里的例子。首先看Kettle数据库资源库 (KettleDatabaseRepository类)。这个资源库可以把Kettle元数据串行化到关系型数据库里。然后还可以看看文件资源库 (KettleFileRepository类)。它在Apache VFS 文件位置上,如磁盘目录、zip文件等,提供了资源库接口层。因为文件资源库相对简单,我们建议你在开发自己的资源库前,先看文件资源库的实现方法。

Pentaho 企业版资源库也使用了资源库插件类型,Pentaho企业版的插件里包括了高级版本管理、文件锁和安全等很多高级特性。实现这些功能,需要投入大量的开发时间。

文件锁这些高级特性没有在Repository接口里直接定义,Pentaho可以通过服务注册的方式扩展标准功能。关于这个主题的内容请参考Pentaho的WiKi: <http://wiki.pentaho.com/display/EAI/Registering+Services+to+the+Repository+in+Kettle>。

最后,资源库类型的插件应该部署到 `plugins/repositories` 目录下,同时在它的类的annotation里要有@RepositoryPlugin 说明。

23.7 数据库类型插件

关系型和列存储数据库都在不断发展变化中,还不断出现新的数据库和数据库的新版本,这样维护一个数据抽象层并不是一个容易的工作。抽象层的目标是使Kettle可以更容易更方便地支持一种新的数据库。过去,Kettle可以很好地支持不同数据库,它可以支持99%的数据库。但这几年,不断出现的数据库新版本,问题不断的数据库驱动,都需要调整Kettle现存的数据库部分的功能架构,所以就出现了数据库类型的插件。

数据库类型插件主要实现org.pentaho.di.core.database.DatabaseInterface接口。这个接口包含了很多描述数据库行为的方法。开发数据库插件时可以根据情况覆盖或定义其中的任何方法。例如,我们创建MySQL 5.1的数据库插件。本章前面讲过,如果把插件的类型ID命名为“MYSQL”,我们创建的插件就会代替Kettle自带的MySQL数据库插件。所以我们不能使用“MYSQL”作为类型ID,这样我们创建的数据库插件在创建数据库连接窗口的数据库列表里才是单独的一项 (在本章的例子中可以找到例子对应的MySQL51DatabaseMeta.java 文件)。

```
package org.kettlesolutions.plugin.database.mysql51db
import org.pentaho.di.core.database.DatabaseInterface;
import org.pentaho.di.core.database.MySQLDatabaseMeta;
```



```
import org.pentaho.di.core.plugins.DatabaseMetaPlugin;
```

```
@DatabaseMetaPlugin(
    type="MYSQL51",
    typeDescription="MySQL 5.1"
)
```

MySQL 5.1 使用了不同的JDBC 驱动类，另外我们还让它支持事务：

```
public class MySQL51DatabaseMeta extends MySQLDatabaseMeta
    implements DatabaseInterface {
```

```
    public String getDriverClass() {
        return "com.mysql.jdbc.Driver";
    }
```

```
    public boolean supportsTransactions() {
        return true;
    }
}
```

部署时，要把这个插件放在 `plugins/databases` 目录下。

在编写本书时，数据库插件的用户界面接口还没有完成。因为数据库对话框是 `core`（或 `common`）`project`，而且图形界面使用了 `XUL`，所以需要在接口的 `getXulOverlayFile()` 方法里指定一个文件名。在这个例子里，我们把文件名设置为“`mysql`”，这样，文件名后面还要加上访问类型，最后的文件名应该是 `mysql_native.xul`。关于插件定义的 `XUL` 文件的规范很快就会出来。目前，可以通过硬编码的方式来代替接口里的值，或者从一个配置文件里读取（可以通过变量的方式）。

23.8 小结

在本章学习了如何写一个 Kettle 插件，我们了解了插件架构和如何搭建开发环境。本章讨论了下面的内容：

- 步骤插件的四个核心接口的实现。
- 处理步骤插件的几个主要问题，如发送数据行到指定步骤、变量替换等。
- 如何写用户自定义 Java 类步骤的代码。
- SWT 用户界面的开发基础。
- 如何创建作业项插件。
- 设计一个自定义分区的方法。
- 构建资源库类型的插件。
- 创建数据库类型的插件。

附录A Kettle生态群

阅读完本书，你会觉得使用Kettle做ETL开发不但简单而且充满乐趣。为了能让工作更简单和更有乐趣，你可能还希望和其他Kettle粉丝交流；或者你在开发过程中遇到了问题时，希望知道这个问题是一个新问题还是一个已知问题。你可能还希望及时获得最新的版本，试用一些新功能，甚至为Kettle社区做贡献。本附录就介绍构成整个Kettle生态群的网站和工具。

Kettle 开发和版本

Kettle 项目的代码库更新很频繁，每天都有更新，修改Bug，增加新的功能，不断的优化。所以，就会有五个Kettle 版本在同时使用。每个版本都位于产品生命期的不同阶段。下面说明这五个版本的不同作用，以及如何选择合适的版本。从GA版本（一般可用版本）到Trunk版本（源代码，需要自己编译）。

- **GA（一般发布）版本：**Kettle 最稳定的版本，可以在生产环境中使用。事实上，在生产环境里，我们只建议使用 GA 版本。GA 版本一般都经过足够的单元或集成测试，被世界上的数千个用户使用。如果从Pentaho 社区下载最新的稳定版本，会进入<http://wiki.pentaho.com/display/COM/Latest+Stable+Builds> 这个链接。你会发现这个页面里的版本可能不是最新的版本，如果想要最新的 GA 版本，最好从 SourceForge 上下载，<http://sourceforge.net/projects/pentaho/files>。
- **RC（Release Candidate）：**候选版本，可能会成为下一个发布版本，一般会有多个候选版本供下载。候选版本有多项作用，首先，企业可以使用候选版本进行升级测试，以确保现有的功能不受影响；其次，ETL开发人员可以试用候选版本里的一些新特性；最后，Pentaho 可以从用户那里得到关于这些新功能的反馈和问题。RC版是不再提交新代码的版本，所以各个RC 版和最终的GA版没有功能性的差异。RC版也可以从 SourceForge 网站上获取。
- **Milestone：**在新版本的开发过程中，会有多个里程碑版本，每个里程碑版本里都会有一些新功能。Pentaho 开发人员喜欢敏捷开发方法，敏捷开发方法从“问题（issue）”开始，每个问题可能是指一个新特性、一项性能改进或者一个bug。每个问题又可以分为多个“冲刺（sprints）”，每个冲刺的最终结果都会体现在里程碑里。这些里程碑有时以快照的形式提供下载，有时也不提供下载。在WiKi 页面上可以找到里程碑版本，<http://wiki.pentaho.com/display/COM/Latest+Milestones>，有时持续集成的站点（见下面部分）上也支持下载。
- **CI（持续集成，Continuous Integration）：**Pentaho 使用 Hudson 做持续集成。从<http://ci.pentaho.com> 可以访问Pentaho 的持续集成环境，在这里可以找到所有Pentaho项目（包括Kettle）的最新的build 版本。这里有不同 Pentaho 项目的二进制版本，但没有任何安装程序，所以要懂得如何手工安装和部署。对 Kettle 来说，安装部署很容易，只要下载最新的 zip 文件（文件名中带版本号和 build 次数的文件，而不是pdi-ce-TRUNK-SNAPSHOT.zip 文件），解压缩到一个目录，打开终端，执行spoon.sh（Linux）或

spoon.bat (Windows) 文件。在Mac系统上, 直接点击 Data Integration 64-bit.app 文件。

- Trunk 版本: 如果一个开发人员想给 Kettle 提交代码, 或者只是希望使用源代码工作, 可以从 Subversion 上下载代码。Kettle 代码的地址是<http://source.pentaho.org/svnkettleroot>。关于代码如何下载和使用, 可以访问页面 <http://community.pentaho.com/getthecode>。(2013年10月份, Kettle代码已经移到了 GitHub上。——译者注)

Pentaho社区wiki

对学习Pentaho的大多数人来说, 获取文档的首选方式就是访问社区wiki, <http://wiki.pentaho.com>。wiki 分为几个部分, 每个 Pentaho 项目都有自己独立的页面, 在这里可以找到文档、常见问题、演示等。尽管看上去文档都应该在这里找到, 但实际却有点让人失望。很多链接指向的都是空页面, 或者是在描述一个早期的版本。对于Kettle, 可能有些步骤根本找不到文档, 要知道wiki里的内容完全由 Pentaho 社区来创建维护。但这也是个好消息, 如果你在社区注册了, 你也可以更新这里的内容, 来帮助其他的用户使用Kettle。

使用论坛

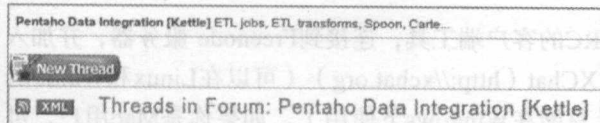
和wiki不同, wiki更静态化, 而论坛交互性更强。在这里Kettle粉丝之间互相交流, 有问题可以寻求其他人的帮助。Pentaho的论坛是<http://forums.pentaho.org>, 对每个人都开放。但有一条黄金规则要记住, 不仅对Kettle论坛, 对所有开源社区的论坛都适用: 要想获取, 必先奉献。另外要有一些耐心和礼貌。

我们可以在论坛里按照主题搜索现有的帖子, 也可以发新帖, 或回复现有的帖子。关于Pentaho社区的很多消息也都是最先通过论坛发布的。如果不注册你也可以使用论坛搜索, 但是不能发新帖和回复别人的帖子。

在你第一次访问 Pentaho论坛的时候, 你可能会发现, Kettle是论坛里访问量最大的一个板块。Kettle 论坛里现在有40 000个帖子, 所以论坛里可能有你要问的问题。所以在发帖子之前, 要搜索论坛, 确定其中没有同样的问题。关于如何使用论坛还有其他规则, Kettle首席架构师Matt Caster, 把这些规则都列举出来, 具体请看: <http://forums.pentaho.org/showthread.php?t=71633>。

还有一个跟踪论坛内容的好方法, 就是使用像 <http://reader.google.com> 这样的RSS阅读器订阅 Pentaho 论坛的 RSS。在 Kettle 论坛主页上点击 RSS 按钮 (在发布新帖的按钮的下方)。如图A-1所示。

另外, 你也可以直接把下面的 URL 直接添加到你的RSS 阅读器里: <http://forums.pentaho.org/external.php?type=RSS&forumids=135>。

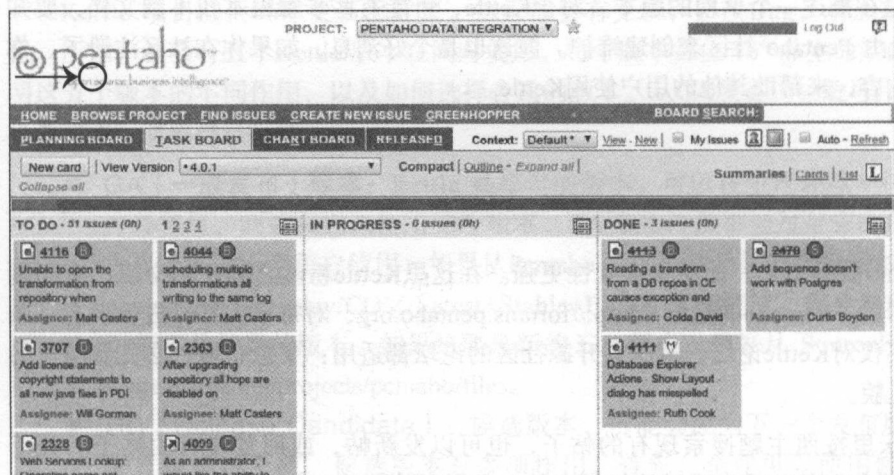


图A-1 订阅论坛 RSS

Jira

Jira是澳大利亚的 Atlassian 公司的产品（<http://www.atlassian.com>），也可能是目前世界上使用最广泛的问题追踪和项目管理系统。很多大公司，如Oracle、Cisco和波音都使用Atlassian 的工具，即使是微软也在使用。所以Jira不是Pentaho的产品，而是Pentaho在使用的产品。因为在社区里大家都知道Jira，Jira就成了一个名称。所以如果你听到谁说“去Jira上找”或者“把它放到Jira”上，实际上就是在说 Pentaho 的问题跟踪系统网站，<http://jira.pentaho.org>。

初看上去，Jira好像是一个bug跟踪系统，但并不完全如此。Jira是一个环境，在这里可以找到Pentaho的发展路线图、开发计划和目前准备发布版本的工作进度，当然也有bug。你可以看到事情的进展情况，问题分配给了谁，问题什么时候可能被解决。图A-2是PDI任务图的例子，左侧显示了51个打开的问题，右侧是三个关闭的问题，没有正在处理的问题。



图A-2 Jira任务仪表盘

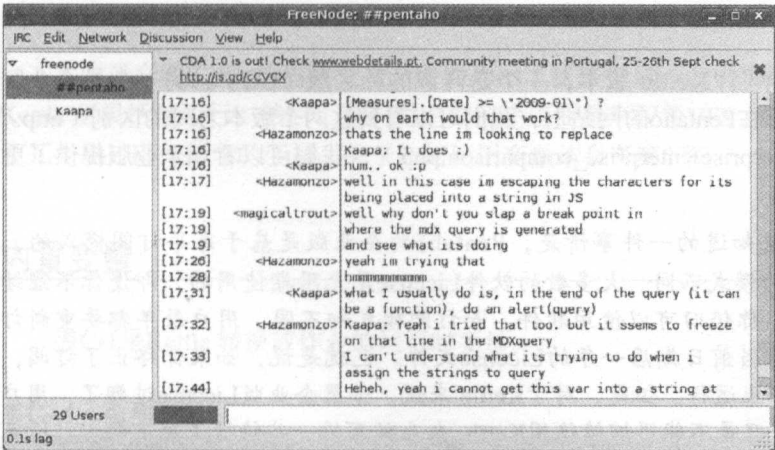
##pentaho

##pentaho 是在IRC（Internet Relay Chat）上的非官方的 Pentaho 频道。IRC是互联网上历史相当悠久的在线聊天工具，有二十多年的历史。它甚至还早于WWW 和 HTML的时代，所以IRC 频道还在使用文本方式，而且是基于客户端/服务器的架构，你需要下载一个客户端才能使用 IRC 连接到各个不同的服务器。每个服务器上你都可以加入一个或多个频道，每个频道通常使用 # 符号开头，但如果你搜索 #pentaho，你不会找到任何结果，因为pentaho使用两个 # 符号开头：##pentaho，这个频道在 Freenode.net 服务器上。

说明：IRC 上有一个通用规则：以一个 # 号开头的频道一般是聊天频道，以两个 # 号开头的频道一般是帮助类的频道。

连接到这个频道上非常容易：下载一个IRC的客户端工具，连接到Freenode 服务器，并加入 ##pentaho 频道。有两个常用的IRC客户端，XChat（<http://xchat.org>）（可以在Linux和Windows下使用）、mIRC（<http://www.mirc.com>）（只能在Windows下使用）。如果你是Mac用户，可以使用XChat Aqua（<http://xchataqua.sourceforge.net>）或者Babbel（<http://www.babbelirc.com>）。

所有这些客户端的使用方式都类似，连上选中的频道以后，直接输入文字即可。图A-3是使用XChat连接到##pentaho频道的图示。



图A-3 使用IRC在 ##pentaho聊天

从图A-3可以看到，使用IRC很简单。从截图中我们可以看到有29个用户在线（这是在星期日），在频道的上方会提示近期的社区聚会。如果和在线用户私聊，这个用户的名字就会出现在左侧的频道列表中。

通常你可以看到25 ~ 40人在这个频道上，很多Pentaho的开发人员在IRC上都很活跃，Pentaho 的社区管理员Doug Moran 也是IRC的常客。图A-1就是很多人在交谈，正在解决一个问题，通常情况下（这里要强调一下：通常）这里的开发人员都很乐于助人。

警告：不要期盼着 IRC 是一个免费技术支持频道，你的问题都会马上被解决。在大多数情况下，你的问题应该放到forum或Jira上，可能你的问题要过几个小时才有人发现（或者根本没人发现）。

在图A-3的这个屏幕截图里，可以看到频道里所有用户的名字。大多数用户使用的别名。大多数用户实际也是Pentaho论坛的用户，如果你经常访问论坛，应该能看出来谁是谁。

附录B Kettle 企业版特性

我们知道Pentaho 提供了两个 Kettle 版本，一个是开源的社区版（CE），另一个是企业版（EE），这是一个商业软件。在Pentaho的网站上，我们可以看到这两个版本之间的区别（http://www.pentaho.com/products/enterprise/enterprise_comparison.php），我们可以看到企业版提供了更多的功能。

说明：关于企业版，需要知道的一件事情是，Pentaho的企业版是基于每年订阅模式的，这种模式和软件License的模式不同。大多数的软件License是无限期使用的，即使你不继续为技术支持和服务付费，你仍旧可以使用软件。而订阅模式却不同，用户每年都要重新订阅，以获得一个有效期为当前日期后一年的License文件。也就是说，如果你终止了订阅，License一过期，软件就停止运行。但是，对于Kettle 来说，如果企业版License过期了，用户还可以继续使用社区版。只是不能再继续使用Kettle 企业版里的一些特性（见下面），所有在企业版里设计的作业或转换都可以在社区版里运行。

在Kettle 4 版本之前，Kettle的企业版和社区版没有太大差别。唯一的差别就是企业版有管理控制台，你可以使用这个控制台调度和监控作业。但从Kettle 4开始，差别开始逐渐增多。尽管Kettle 4 社区版仍然是一个强大的工具，它提供了很多商业ETL工具才有的功能，但和Kettle企业版相比，少了如下的一些功能。

- **安装程序：**尽管安装Kettle很容易，但企业版还是提供了一个安装程序，这样更简化了安装过程。
- **敏捷BI：**Kettle企业版的安装程序里有敏捷BI的插件（见第11章）。而Kettle社区版的用户需要自己下载、解压和部署。
- **集成的调度器：**Kettle企业版有内置的调度工具，不用再使用Pentaho BA的控制台，也不用再使用一些第三方的调度工具。
- **数据整合服务器：**Kettle EE还有一个数据整合服务器，这个服务器独立于Pentaho的其他组件，也使得Kettle企业版容易部署和管理。
- **管理控制台：**Pentaho的企业版控制台也可用于PDI的管理控制台。
- **企业版资源库：**CE和EE最大的区别可能就是和数据整合服务器紧密工作的企业版资源库。企业版资源库的功能是Kettle社区版以前没有的，它面向于有多个开发人员的开发团队，并提供了下面一些功能。
 - **版本控制：**可以在资源库里保存多个版本的作业和转换，并可以回滚到前一个版本。
 - **锁：**通过“检入/检出”功能阻止开发人员覆盖其他人的工作。
 - **安全性：**支持细粒度的用户授权，可以建立用户、角色、权限。默认情况下，资源库的安全机制和Pentaho 的安全机制是结合在一起的，但也可以和已有的一些安全或授权机制结合起来，如通过LDAP 和 Active Directory。
- **其他步骤：**企业版里还有很多社区版里没有的步骤，如Google分析输入步骤、Google文档输入步骤、JMS步骤，等等。

订阅企业版，用户可以免费获得电话和电子邮件支持，还包括很多的参考案例。即使你不想购买 Kettle企业版，你也可以购买Kettle社区版的技术支持，在编写本书时，Pentaho也为Kettle社区版提供了收费的技术支持服务。

附录C 内置的变量和属性参考

本附录介绍Kettle的内置变量。下面列出了这些变量，我们可以在运行时使用这些变量，使Kettle 更灵活地运行。然后，我们讨论如何使用变量来配置VFS，进行SFTP验证，另外我们还介绍几个JRE 运行环境的变量。关于如何使用变量请参考第2章。

内置变量

表C-1是Kettle 转换或作业的运行时变量。

表C-1 内部变量

| 变量 | 描述 |
|--|--|
| Internal.Kettle.Version | 这是Kettle的版本号，例如4.0.0 |
| Internal.Kettle.Build.Version | 这是Kettle源代码的SVN的修订号 |
| Internal.Kettle.Build.Date | 这是Kettle的build日期 |
| Internal.Job.Filename.Directory | 如果使用文件方式运行作业（.kjb），这个变量就是作业文件所在的目录。利用这个变量用户可以指定其他的文件 |
| Internal.Job.Filename.Name | 如果使用文件方式运行作业（.kjb），这个变量就是作业文件名 |
| Internal.Job.Name | 当前正在执行的作业的名字 |
| Internal.Job.Repository.Directory | 如果使用资源库方式运行作业，这个变量是作业所在资源库目录的路径 |
| Internal.Transformation.Filename.Directory | 如果使用文件方式运行转换（.ktr），这个变量就是转换文件所在的目录。利用这个变量用户可以指定其他的文件 |
| Internal.Transformation.Filename.Name | 如果使用文件方式运行转换（.ktr），这个变量就是转换文件名 |
| Internal.Transformation.Name | 当前正在执行的转换的名字 |
| Internal.Transformation.Repository.Directory | 如果使用资源库方式运行转换，这个变量是转换所在资源库目录的路径 |
| Internal.Step.Partition.ID | 如果一个步骤是以分区方式运行的，每个分区都有一个步骤拷贝。这个变量就是步骤拷贝所属的分区ID |
| Internal.Step.Partition.Number | 如果一个步骤是以分区方式运行的，每个分区都有一个步骤拷贝。这个变量就是步骤拷贝所属的分区编号，分区编号从0到分区个数减1 |
| Internal.Slave.Transformation.Number | 如果转换是在子服务器上以集群方式运行的（不是主服务器），这个变量就是子服务器的编号，编号从0到子服务器个数减1 |
| Internal.Slave.Server.Name | 如果转换在子服务器上以集群方式运行，这个变量就是子服务器的名字 |
| Internal.Cluster.Size | 如果转换在子服务器上以集群方式运行，这个变量就是集群中子服务器的个数 |

续表

| 变量 | 描述 |
|-----------------------------|--|
| Internal.Step.Unique.Number | 这个变量是指定步骤的步骤拷贝的唯一编号。这个变量同样适用于分区和集群环境。取值从0到步骤拷贝个数减1 |
| Internal.Cluster.Master | 若转换以集群方式运行，如果是运行在主服务器上，这个值是Y，如果是运行在子服务器上，这个值是N |
| Internal.Step.Unique.Count | 唯一的步骤拷贝个数。也适用于集群或分区的情况 |
| Internal.Step.Name | 正在执行的步骤的名字 |
| Internal.Step.CopyNr | 本地转换的步骤拷贝号（不考虑集群的情况） |

Kettle 变量

表C-2是一组变量，用于配置与Kettle 运行相关的一些配置项。

表C-2 Kettle变量

| 变量 | 描述 |
|--|--|
| KETTLE_SHARED_OBJECTS | 作业和转换的共享对象文件的位置。默认的共享对象文件shared.xml，位于Kettle home 目录下。设置这个变量可以覆盖默认值 |
| KETTLE_EMPTY_STRING_DIFFERS_FROM_NULL | 如果这个变量设置为Y，空字符串和null是不同的，否则就是相同的（默认） |
| KETTLE_MAX_LOG_SIZE_IN_LINES | Kettle初始的最大日志行数。设置为0将保留所有日志行（默认） |
| KETTLE_MAX_LOG_TIMEOUT_IN_MINUTES | Kettle中日志行的最长保留时间（单位：分钟）。设置为0将保留所有行（默认） |
| KETTLE_STEP_PERFORMANCE_SNAPSHOT_LIMIT | 内存中的最大步骤性能快照数。设置为0将保留所有快照（默认） |
| KETTLE_PLUGIN_CLASSES | 逗号分隔的类名列表，用来查找插件的annotation。参考 http://wiki.pentaho.com/display/EAI/How+to+debug+a+Kettle+4+plugin |
| KETTLE_LOG_SIZE_LIMIT | 如果在转换和作业的日志表的配置中没有设置“日志行数限制”参数，这类转换和作业将统一使用这个参数作为最大日志行 |

另外在表C-3中还列出了几个变量，使用这些变量可以为所有转换自动化地配置不同的日志表。使用前，用下面的名字替换变量名中的“...”。

- TRANS：对于转换日志表。
- TRANS_PERFORMANCE：对于性能日志表。
- STEP：对于步骤日志表。
- JOB：对于作业日志表。
- JOBENTRY：对于作业项目日志表。
- CHANNEL：对于通道日志表。

关于日志表的更多内容请参考第14章。

表C-3 Kettle日志表变量

| 变量 | 描述 |
|-----------------------|------------------|
| KETTLE_..._LOG_DB | 设置日志表使用的数据库连接的名字 |
| KETTLE_..._LOG_SCHEMA | 设置日志表使用的数据库模式名 |
| KETTLE_..._LOG_TABLE | 设置日志表使用的日志表名 |

配置VFS需要的变量

在第2章曾经介绍过，我们可以使用URI来指定文件的位置。这里使用的Apache VFS 技术支持很多不同的文件系统，例如file://、zip://、ftp://、http://等。对于这些文件系统，通常标准的验证方式是在主机名前加上username@password，如：

```
ftp://john:pwd4john@example.com/pub/customer/file.txt
```

对于大多数文件系统，这种方法都可以正常工作。但是，对于安全FTP（sftp://），需要指定额外的一些信息，例如私钥文件的位置。通过变量可以满足这个要求，但需要遵守下面的变量命名规则。

- 变量名以vfs 开头。
- 后面是文件系统的模式名：sftp。
- 后面是你设置的参数。
 - StrictHostKeyChecking：如果设置为no，可以接受任何远程主机的证书。如果设置为yes，远程主机必须在hosts 文件中（~/ssh/known_hosts）。
 - authkeyphrase：私钥文件可能需要的密码。
 - identity：私钥文件路径，用于SFTP的验证。
- 最后，变量名的后面要跟着主机名或IP地址，变量会在这个主机上被创建。

参考下面的几个例子：

- vfs.sftp.StrictHostKeyChecking.sftp.myhost.net
- vfs.sftp.authkeyphrase.sftp.example.com
- vfs.sftp.identity.192.168.1.5

说明：关于配置 Kettle VFS 的更多内容请参考 <http://wiki.pentaho.com/display/COM/Configuring+Kettle+VFS>。

要注意的 JRE 变量

如果要获得全部的 Java 运行环境变量，需要使用 System 类的getProperties()方法。[http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html#getProperties\(\)](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html#getProperties())。表C-4 列出了几个在转换和作业中可能会用到的 JRE 变量。

表C-4 要注意的 JRE 变量

| 变量 | 描述 |
|----------------|--|
| java.version | JRE 版本号 |
| java.io.tmpdir | 默认的临时文件目录。Kettle产生的所有临时文件和Spoon日志文件都保存在这个目录下 |
| os.name | 操作系统的名字 |
| os.arch | 操作系统的架构 |
| os.version | 操作系统版本号 |
| user.name | 用户名 |
| user.home | 用户的 home 目录 |
| user.dir | 当前工作路径 |

好书分享



《海量运维、运营规划之道》

ISBN: 978-7-121-21796-8



《收获，不止Oracle》

ISBN: 978-7-121-20070-0

Pentaho Kettle 解决方案： 使用PDI构建开源ETL解决方案

Kettle是一款功能和性能都可扩展的开源ETL和数据整合工具，使用Kettle你可以从数据库、平面文件、XML文件、Web 服务、ERP系统和OLAP立方体中抽取数据。它提供了超过120个内置的转换步骤来检验、清洗和确认数据，另外还有很多选项用来把数据加载到数据仓库和其他目标系统。相对于传统的商业软件如Informatica PowerCenter、IBM InfoSphere DataStage和BusinessObjects Data Integrator来说，Kettle是一个易于使用的、低成本的解决方案。

本书详细介绍了如何使用Kettle创建、测试和部署ETL和数据整合解决方案。通过本书，读者可以学会如何用Kettle来创建转换和作业，如何进行版本控制，审计数据和进行调度。读者可以进一步了解到一些更深入的主题，如集群和云计算、实时数据整合、加载Data Vault模型和开发Kettle插件。另外，本书还提供了一些案例，通过这些案例，读者可以了解如何在实际中应用Kettle。

- ◎探索Kettle ETL工具集里的组件。
- ◎学习如何安装和配置Kettle，以连接到不同的数据源和目标。
- ◎学习使用Kettle设计和构建ETL解决方案中的每个环节。
- ◎学习如何使用Kettle加载数据仓库。
- ◎学习部署和调度ETL解决方案的步骤。
- ◎学习如何把Kettle集成到第三方产品中。
- ◎学习如何扩展Kettle和开发插件。
- ◎学习如何使用集群和云计算来提高ETL解决方案的性能。
- ◎学习如何使用Kettle来做实时数据整合。

WILEY



新浪微博
weibo.com

@博文视点Broadview



上架建议：计算机\商业智能

ISBN 978-7-121-22445-4



9 787121 224454 >

定价：89.00元



策划编辑：张月萍

责任编辑：贾莉

封面设计：李玲